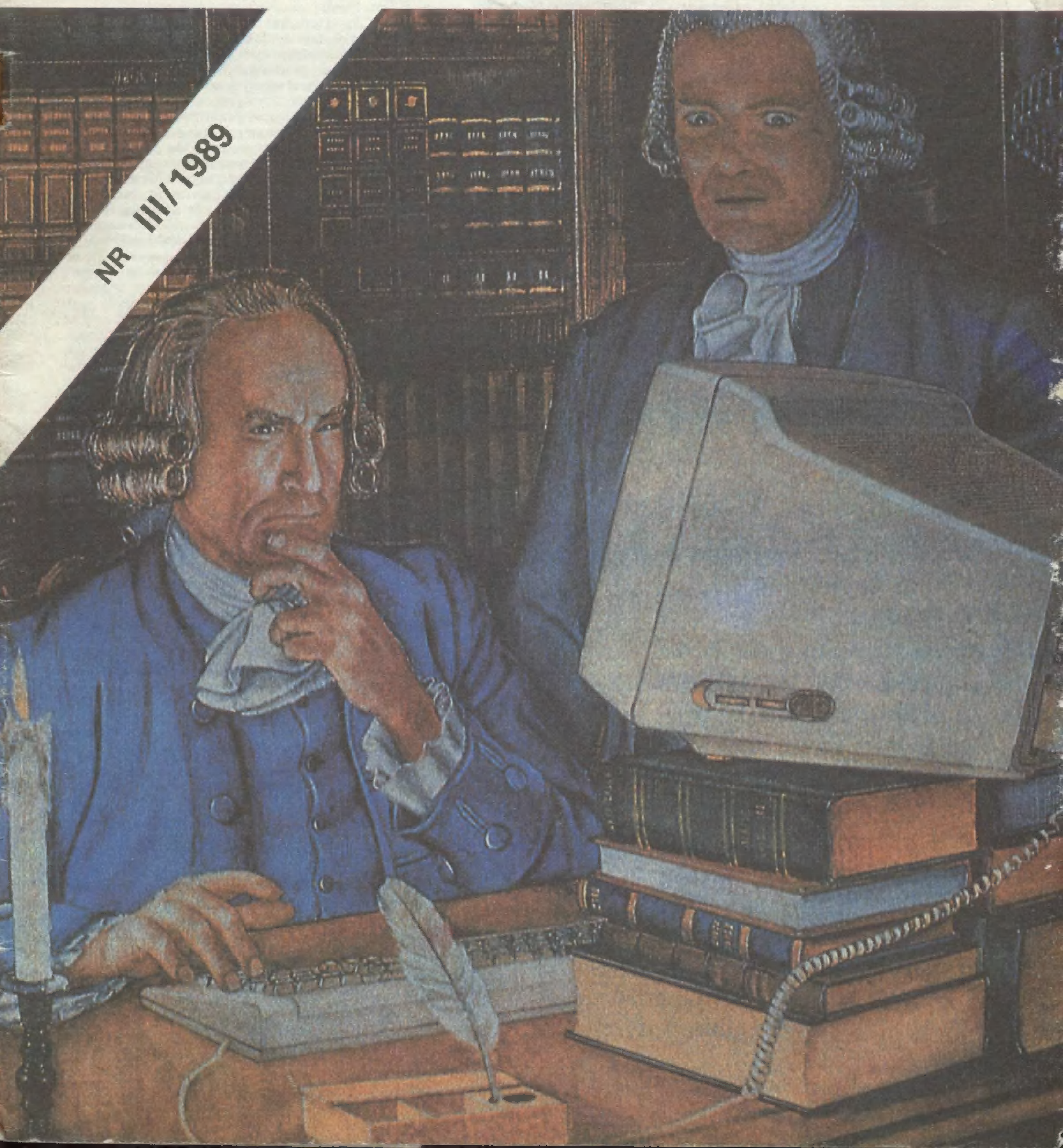
młody TECHNIK

InforMik

MAGAZYN KOMPUTEROWY „MŁODEGO TECHNIKA"



Nasz komentarz:

O POTRZEBIE ŁADU I PORZĄDKU

Drogi Czytelniku, błagam! Nie sugeruj się tytułem! Ten felieton bynajmniej nie jest dedykowany wiadomym czynnikom: jak zwykle zamierzam się bowiem pastwić nad (mikro) informatyką, głównie rodzimą.

Pasjonując się na bieżąco rozwojem techniki mikrokomputerowej, dobrze jest nie tracić historycznej perspektywy. Jeśli idzie o sam sprzęt, to nawiązując do historii motoryzacji można by założyć, że jesteśmy w okresie ekspansji słynnego Forda T, którego masowa produkcja spowodowała skutki zbliżone do ekspansji IBM PC i jego pociotków. Jeśli natomiast rozpatrzymy komputer osobisty po prostu jako narzędzie pracy, „czarną skrzynkę", to sądzę, że odniesieniem w historii automobilizmu musiałyby być czasy dużo wcześniejsze. Zapewne byłby to okres pionierskich pojazdów budowanych na podobieństwo bryczek, pokracznych dziwolągów plujących ogniem i smrodem.

Motoryzacja dawno przestała być prywatną sprawą konstruktorów. Dziś każdy wie, jak z grubsza powinien wyglądać samochód, a zwłaszcza, jak powinien być obsługiwany. Biada projektantowi, który odrzucając kierownicę zaproponowałby np. zmianę kierunku za pomocą pedałów, a dźwignie hamulca głównego i sprzęgła wetknął kierowcy w ręce. Klienci tego nie zaakceptują, chociażby wytoczyć tysiąc argumentów. Co innego przed 90 laty. Każdy mógł tak zmajstrować organa kierujące, jak mu się żywnie podobało i nikomu to nie przeszkadzało. W końcu jedynym szoferem bywał właściciel-konstruktor.

Zasady obsługi samochodów są dzisiaj dalece ujednolicone, tak że kierowca „malucha" bez kłopotu opanuje jazdę mercedesem i odwrotnie. W każdym razie podstawowe przyzwyczajenia kierowcy zostaną uszanowane. A jak to wygląda z oprogramowaniem? Jeszcze przed kilku laty każdy producent organizował dialog programu z użytkownikiem w sposób zupełnie dowolny, wykoncypowany samodzielnie lub podpatrzony u konkurencji. W każdym razie obsługa konkretnego programu bywała sztuką samą w sobie. Zaczynając pracę z nowym programem należało zdobywać umiejętności od zera, a dotychczasowe doświadczenie raczej przeszkadzało niż pomagało (stare nawyki!).

Czy doczekamy błogosławionych czasów, w których zasady obsługi typowych programów zostaną ujednolicone tak, jak technika prowadzenia auta? Wagę problemu doceniła m.in. firma IBM, lansując system jednolitej architektury oprogramowania, który ma obejmować pełne spektrum sprzętu: od komputerów osobistych do superkomputerów. Chodzi m.in. o to, aby niezależnie od tego, przed jakim ekranem posadzimy użytkownika, od razu poczuł się on swojsko w znajomym środowisku.

Innym przedsięwzięciem, zmierzającym do ujednolicenia dialogu człowieka z komputerem, jest system Microsoft Windows. Wszystkie programy, tworzone jako jego aplikacje, korzystają z tych samych podstawowych środków dialogu, jak rozwijane w dół menu, okna dialogowe, przewijające się listy itd. Dość, że człowiek nabędzie wprawy w obsłudze jednej aplikacji, to do uruchomienia następnej i wykonania w niej podstawowych czynności podręcznik jest zbędny. W każdym razie nie trzeba uczyć się nowych technik dialogu. Wystarczy przeczytać, co program robi, a jak wywołać daną czynność, często podpowie intuicja. Twórcy różnych aplikacji Windows starają się np. możliwie jednolicie rozmieszczać poszczególne opcje menu, a nawet w miarę możliwości stosować typowe nazwy standardowych opcji. Wszystko to służy powiększeniu psychicznego komfortu użytkownika i zminimalizowaniu czasu, potrzebnego na opanowanie nowego programu. W krajach, gdzie czas ma wysoką cenę, jest to argument o poważnym znaczeniu.

Systemy operacyjne OS-2 czy DOS 4.0 są przykładem dążenia do takiej właśnie jednolitości dialogu. Ale nawet producenci nie korzystający z systemu Windows starają się często stosować w swych produktach zapożyczone z Windows techniki dialogu. W Polsce jesteśmy ciągle dalecy od tak wysublimowanych problemów. Nie możemy uporać się ze sprawami tak elementarnymi, jak chociażby ujednolicenie terminologii informatycznej, albo sposobu kodowania polskich liter. Czyżby nie było nikogo predestynowanego do takiego przedsięwzięcia? Instytucje państwowe — to w czasach mody na demokrację nie brzmi dobrze. Co na to jednak np. Polskie Towarzystwo Informatyczne, które nie przepuszcza żadnej okazji do biadolenia nad kalekim stanem komputeryzacji, a także nad sobiepaństwem amatorów, rzemieślników, spółek itd., które robią swoje nie oglądając się na autorytety?

Niestety, w tak skomercjalizowanej i urynkowionej dziedzinie jak mikroinformatyka siły oddziaływania nie zdobywa się przez wymądrzanie się. Siła ta musi wynikać z faktów dokonanych o wyraźnej, jednoznacznej wymowie. Za takie trudno jednak uznać gremialne nasiadówki albo firmowanie chałtur, realizowanych przez ludzi, którym jest obojętne, pod jakim szyldem pracują, byle szyld dobrze wyglądał i pozwalał wynegocjować tłustą zapłatę.

Uporządkowanie i ujednolicenie zasad obsługi szeroko rozumianych systemów komputerowych staje się problemem coraz lepiej postrzeganym, w miarę jak rola komputera w naszej cywilizacji staje się porównywalna z rolą samochodu. Warto już teraz poczynić praktyczne kroki. Jednym z najistotniejszych jest oddziaływanie na twórców oprogramowania, podsuwanie im odpowiednich wzorców oraz aparatu pojęciowego i metodycznego, a także narzędzi. Oprócz tego trzeba stworzyć obowiązujące, choć niekoniecznie formalne, zasady nazewnictwa typowych obiektów i czynności, tworzenia komunikatów i meldunków. Chodzi o to, aby te same rzeczy lub zdarzenia nazywały się tak samo w różnych programach. Nie zawsze można wymagać od programisty trafnych propozycji językowych. Czasem należałoby mu trochę pomóc. Czy opracowanie i rozpropagowanie np. krajowych wytycznych dotyczących terminologii informatycznej i zasad dialogu z użytkownikiem w języku polskim nie byłoby zadaniem godnym PTI?

Istotne jest też uświadamianie programistom wagi problemu, gdyż wielu nawet dobrych programistów odnosi się do problemów dialogu z użytkownikiem z zaskakującym lekceważeniem i z niechęcią do opierania się na rozpowszechnionych konwencjach. Nie wszyscy muszą przecież sami odkrywać Amerykę — większość może tam dotrzeć po śladach Kolumba. Aby pójść utartym, dobrze wytyczonym szlakiem, trzeba mocnych i budzących zaufanie drogowskazów. Ktoś mocny powinien je jednak powbijać.

Roland Wacławek

„Młody Technik — InforMik" wydaje Instytut Wydawniczy „Nasza Księgarnia"

Rada Redakcyjna: doc. dr Zygmunt Dąbrowski, inż. Jerzy Jasiuk, dr Zygmunt Kalisz, mgr Zbigniew Słowiński, mgr inż. Jerzy Siek, dr Zbigniew Płochocki, Piotr Postawka, mgr inż. Roland Wacławek, prof. dr hab. Andrzej K. Wróblewski (przewodniczący), mgr inż. Grzegorz Zalot.

Zespół redakcyjny: *„InforMik"* redaguje zespół *„Młodego Technika"*. Jerzy Klawiński (sekretarz red.), Jacek Nowicki (red.), Dariusz A. Przygoda (red.), Lidia Sadowska-Szlaga (korekta), Józef Trzionka (redaktor naczelny), Roland Wacławek (software), Grzegorz Zalot (hardware), Izabella Żur (red. tech.).

Stali współpracownicy: Wojciech Apel, Tadeusz Basista, Jacek Jędrzejowski, Piotr Postawka, Marek Szczepański, Krzysztof Wiśniewski.

Adres redakcji: ul. Spasowskiego 4, 00-389 Warszawa, lub skr. poczt. 380, 00-950 Warszawa. **Telefony:** centrala: 26 24 31 do 36. Dział Łączności z Czytelnikami — wewn. 60, pozostałe działy: wewn. 42 i 47. Redaktor naczelny: 26 26 27 lub wewn. 87.

Warunki prenumeraty: ogólnie obowiązujące w kraju. **W STAŁEJ SPRZEDAŻY INFORMIK JEST W SALONIE WYDAWNICZYM „NASZEJ KSIĘGARNI"** ul. Spasowskiego 4 A.

Druk: Zakłady Graficzne w Katowicach. Zam. 0986/4333/9
Nakład 50 315 egz. A-66

młody TECHNIK

InforMik

MAGAZYN KOMPUTEROWY
„MŁODEGO TECHNIKA"
NR III [11] ROCZNIK III

SPIS TREŚCI

Zdjęcia w numerze: Władysław P. Jabłoński, „Byte", „Robotron", „Profil", ze zbiorów redakcji.

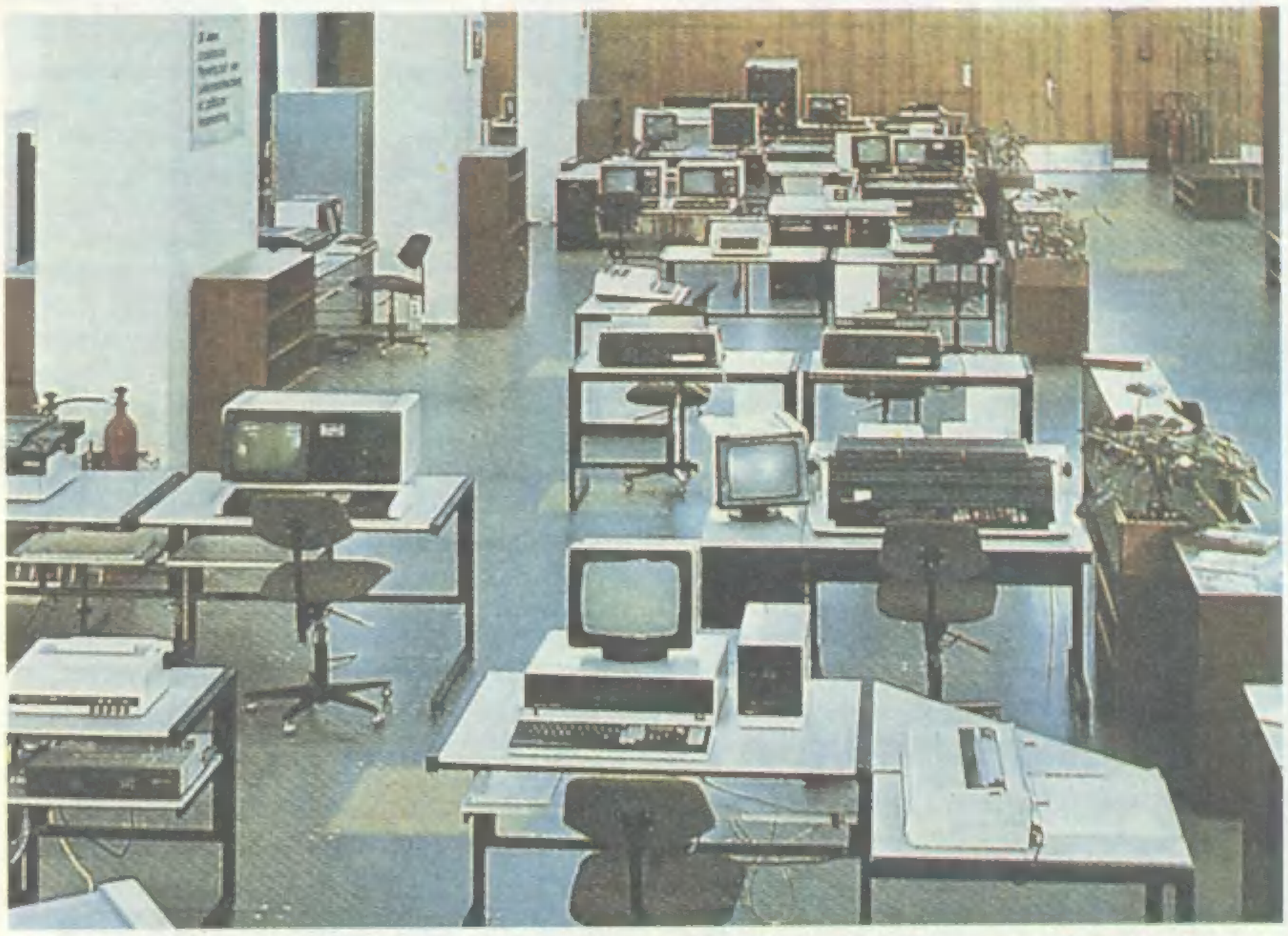
Myśli z ogłoszenia

Wprawdzie nie podniecam się tak czytaniem anonsów prasowych jak niektórzy spośród moich redakcyjnych Kolegów, ale czasem znajdzie się jakiś smakowity kąsek i dla kogoś, kto nie pasjonuje się wymianą M-4 w Gdańsku na podobne w Warszawie lub odwrotnie. Mój Przyjaciel np. przywiózł mi z Indii miejscowe gazety angielskojęzyczne z cudownymi „perełkami" sztuki ogłoszeniowej. Oj, uczmy się od Hindusów metody reklamowania swoich możliwości! Nie każda bowiem dziewczyna w Polsce dając ogłoszenie matrymonialne określiłaby się jako „extremely beauty" (w wolnym przekładzie: wyjątkowa piękność) i nie każdy kamieniarz z Bródna lub Powązek wyzywałby poprzez ogłoszenie prasowe wszystkich kamieniarzy świata do walki w swej profesji jak pewien mistrz z dorzecza Indusu lub Gangesu. Oczywiście w Indiach ogłoszenia matrymonialne zamieszczają rodzice kandydata lub kandydatki, a kamieniarstwo nie ogranicza się do nagrobków i kominków wykonywanych z niezbyt urozmaiconych surowców krajowych, ale agresywność takiego ogłoszenia jest dla Polaka i tak nieco szokująca.

Jak się to wszystko ma do komputera? Nie, nie, nie będę po raz kolejny pisał o prymitywizmie naszej rodzimej reklamy czy o dziesiątkach ogłoszeń firm komputerowych i pseudokomputerowych zajmujących niestety coraz więcej miejsca w merytorycznych częściach pism ze szkodą dla Czytelnika. Zastanawia mnie natomiast nietaktowna z punktu widzenia **etyki** *(już widzę, jak handlowcy pukają się wymownie w mądre, pobrużdżone w mozole nad zdobywaniem dolara i złotówki, czoła) próba przekonania Czytelnika, że tu się sprzedaje coś zupełnie nadzwyczajnego i nie spotykanego u tej wrednej konkurencji. Jak bowiem wszyscy nieco lepiej zorientowani wiedzą, Polaków na najdroższy i najlepszy (a te dwie rzeczy niestety się łączą!) sprzęt nie stać, a poza tym przepisy COCOM-u znacznie utrudniają jego import do krajów socjalistycznych. Tak więc na naszym rynku dominuje tajwańsko-koreański „klon" IBM-a w różnych obudowach, co dla laików ma ogromne znaczenie (zwłaszcza dla Pań — „O, ten by mi pasował do tej perłowej sukni i moich włoskich pantofli!"). W związku więc z sytuacją panującą na rynku komputerowym uważam, że jesteśmy w pompatyczności naszych reklam i anonsów wcale nie gorsi od rodziców panienki, którą trzeba sprzedawać przez ogłoszenie w gazecie pod szyldem „extremely beauty", bo do tej pory tego (hm, przepraszam za przypadkowe podobieństwo ze slangiem młodzieżowym) towaru nikt nie wypatrzył i nie kupił. Bywa czasem bowiem skrajne piękno tak skrzętnie ukryte, że aż niewidoczne i wtedy najlepiej posunąć się do reklamy. My to też robimy — może w nieco innym — hm — asortymencie.*

Pozostaje jeszcze pytanie o co chodzi autorowi niniejszego tekstu — czy np. nie ma zamiaru niszczyć rodzimych kiełków komputeryzacji? O nie — chodzi tylko o to, by zwyciężyli — przy mniej więcej równym poziomie oferowanego sprzętu — ci najsolidniejsi i najtańsi, a podobno dobry towar nie wymaga reklamy — co najwyżej rzetelnej i fachowej informacji, czego wszystkim użytkownikom komputerów serdecznie życzy niżej podpisany.

JERZY KLAWIŃSKI



ROLAND WACŁAWEK

KTO NIE LUBI PRZETWARZANIA WSADOWEGO?

Podobnie jak praktycznie wszystkie współczesne systemy operacyjne, także MS/PC-DOS dysponuje językiem tzw. przetwarzania wsadowego. Podczas gdy inne języki programowania służą do realizacji, np. obliczeń lub administrowania danymi, to język przetwarzania wsadowego służy do programowania pracy samego systemu operacyjnego. Zamiast mozolnie wprowadzać z klawiatury ciągi tych samych zleceń, można zapisać je w tzw. pliku przetwarzania wsadowego (pliku sterującym przetwarzaniem wsadowym, odtąd zwanym krótko: plikiem wsadowym). Po wywołaniu takiego pliku, wszystkie zawarte w nim zlecenia będą kolejno realizowane, zupełnie tak, jak gdyby były wprowadzane z klawiatury w trybie interakcyjnym. Pliki wsadowe są znakomitym narzędziem do usprawnienia dialogu z systemem operacyjnym, zwłaszcza zaś do automatyzacji czynności rutynowych. Niestety, praktyka dowodzi, że wielu użytkowników komputerów klasy PC/XT i AT znajomość z przetwarzaniem wsadowym ogranicza co najwyżej do pliku AUTOEXEC.BAT, zawierającego kilka sekwencyjnych zleceń. Tymczasem możliwości programowania wsadowego są znacznie szersze i obejmują m.in. przekazywanie parametrów, operacje warunkowe oraz pętle.

W systemie MS/PC-DOS pliki wsadowe noszą standardowe rozszerzenie: **.BAT**. Plik wsadowy to zwykły plik tekstowy, w którym każda linia zawiera pojedyncze zlecenie. Plik taki można utworzyć każdym edytorem dostarczającym tekstu nie sformatowanego, jak np. EDLIN lub SideKick, a nawet zapisać go na dysk wprost z konsoli, podając zlecenie:

```
COPY CON: Dysk:ściezka_do_katalogu\nazwa_piku.BAT
```

Oznaczenie dysku i ścieżki można opuścić, nie wolno natomiast pominąć rozszerzenia .BAT. Po wprowadzeniu ciągu zleceń wystarczy nacisnąć klawisz **(F6)**, lub kombinację **(Ctrl)**+**(Z)**, co powoduje wprowadzenie znaku końca pliku EOF, a potem (Enter). Nazwa: AUTOEXEC.BAT jest zarezerwowana dla pliku wsadowego, którego system operacyjny poszukuje automatycznie natychmiast po załadowaniu i w razie odnalezienia go — przystępuje do realizacji zawartych w nim zleceń jeszcze przed przystąpieniem do konwersacji z operatorem.

Uruchomienie pliku wsadowego odbywa się po prostu przez wprowadzenie z klawiatury jego nazwy, ewentualnie poprzedzonej oznaczeniem stacji dysków i ścieżki i uzupełnionej parametrami, jeśli takie są wymagane. Trzeba jednak pamiętać, że jeśli w tym samym katalogu znajduje się obok pliku typu **.BAT** także plik typu **.EXE** lub **.COM** o identycznej nazwie, to pierwszeństwo w wykonaniu będą miały pliki **.COM** i **.EXE**.

Przykładem zastosowania pliku wsadowego mogą być prace programistyczne, np. w języku asemblera, w któ-

rych stale powtarza się ten sam tok postępowania: edycja programu, uruchomienie asemblacji, jeśli nie wykryto błędów — konsolidacja programu, często, przetwarzanie programu z postaci .EXE na .COM programem EXE2BIN, a na koniec uruchamianie programu pod kontrolą debuggera DEBUG. Ponieważ podczas prac nad programem ten ciąg czynności powtarza się dziesiątki i setki razy, to warto zapisać odpowiednie zlecenie w pliku wsadowym, a potem powierzyć mu automatyczną realizację całego procesu od zakończenia edycji do uruchomienia programu.

W pliku wsadowym mogą wystąpić praktycznie wszystkie zlecenia, używane normalnie w trybie bezpośrednim. Oprócz tego plik wsadowy może zawierać instrukcje sterujące przetwarzaniem wsadowym, pozwalające tworzyć proste programy, realizujące pewne czynności cyklicznie lub wielokrotnie dla różnych argumentów, albo też uzależniające wykonanie zlecenia od spełnienia pewnych warunków. W pliku wsadowym może wystąpić do 10, a w pewnych warunkach i więcej parametrów formalnych, na ogół stosowanych w zleceniach w miejsce nazw plików. Przy wywoływaniu pliku wsadowego można natomiast podać zbiór parametrów rzeczywistych, które podczas realizacji poszczególnych zleceń zostaną wstawione w miejsce odpowiednich parametrów formalnych.

Przypuśćmy, że często zachodzi potrzeba wykonywania następujących operacji: kopiowania wszystkich plików z rozszerzeniem .PAS z dysku A na B, następnie kasowania na dysku A wszystkich plików z rozszerzeniem .BAK. Na koniec należy wyprowadzić, strona po stronie i porządku alfabetycznym, wszystkie pliki z katalogu głównego dysku A. Sporządzimy krótki plik wsadowy, zawierający trzy zlecenia. Zapiszemy go wprost z konsoli na dysku A, np. pod nazwą OCZYSCA.BAT. Oto protokół konwersacji z systemem na ekranie monitora:

```
A> COPY CON: OCZYSCA.BAT
COPY A:*.PAS B:
DEL  A:*.BAK
DIR|SORT|MORE
^Z
A> OCZYSCA
```

Ostatnia linia jest już wywołaniem przygotowanego pliku wsadowego (funkcjonowanie programu wsadowego wymaga obecności na dysku A także systemowych programów SORT i MORE).

Wspomnieliśmy już, że w pliku wsadowym można użyć parametrów formalnych, które w fazie realizacji będą zastąpione parametrami rzeczywistymi, podanymi w linii zlecenia po nazwie pliku wsadowego. Parametry formalne noszą nazwy, złożone ze znaku **'%'** i cyfry **'0'..'9'**, np. %1, %5, itd. Wynika stąd, że w pliku wsadowym można użyć do 10 parametrów formalnych. Liczba parametrów rzeczywistych może być natomiast większa w razie użycia instrukcji **SHIFT**, o czym później. Przy wywołaniu pliku wsadowego, w miejsce parametru 0 zostanie podstawiona nazwa samego pliku wsadowego, zaś w miejsce parametrów 1..9 — kolejne następne parametry rzeczywiste. Oto przykład pliku wsadowego z parametrami formalnymi:

```
COPY %1.BAK %2.PAS
TYPE %2.PAS
TYPE %0.BAT
```

Niech plik nosi nazwę KOPIUJ.BAT. Aby rozpocząć przetwarzanie należy wprowadzić nazwę pliku wsadowego bez rozszerzenia, umieszczając po nazwie kolejne parametry, rozdzielone spacjami:

KOPIUJ PROGRAM TEST

W efekcie zostaną wykonane zlecenia następujące:

```
COPY PROGRAM.BAK TEST.PAS
TYPE TEST.PAS
TYPE KOPIUJ.BAT
```

Po utworzeniu kopii pliku PROGRAM.BAK pod nazwą TEST.PAS, treść pliku TEST.PAS zostanie wylistowana na ekranie. Po niej pojawi się listing zawartości — zawartość samego pliku wsadowego, tzn. trzy przedstawione wcześniej zlecenia z ich parametrami. Jak widać, przy przekazywaniu parametrów można składać z fragmentów dowolne napisy, w tym specyfikacje plików.

Z uwagi na zastosowanie znaku '%' jako przedrostka nazwy parametru formalnego, stosując w pliku wsadowym znak należy go zapisać w postaci podwojonej '%%' (warto zwrócić na to uwagę w instrukcji **FOR**).

Przedstawimy teraz poszczególne instrukcje przetwarzania wsadowego, zaczynając od najprostszych.

ECHO ON lub **ECHO OFF** lub **ECHO komunikat**

Zlecenie: ECHO zmienia na przeciwny stan wewnętrznego przełącznika systemu operacyjnego, sterującego wyprowadzaniem na ekran komunikatów i zleceń pliku wsadowego. Pierwotnie przełącznik echa jest włączony, co powoduje wyprowadzanie na ekran treści wszystkich realizowanych linii pliku wsadowego, tak, jak gdyby były one wprowadzane z klawiatury. Wyprowadzane są też ewentualne komunikaty podane w zleceniach **REM** i **PAUSE**. Po wyłączeniu echa zleceniem: **ECHO OFF**, na ekranie będą pojawiać się tylko teksty wyprowadzane przez poszczególne programy, natomiast treść zleceń pliku wsadowego **.BAT** i komunikaty umieszczone w instrukcjach **REM** i **PAUSE** nie ukażą się (**PAUSE** wyświetli tylko napis: *Strike any key...*). Ponowne uaktywnienie (włączenie) echa jest możliwe zleceniem: **ECHO ON**.

Jeśli plik wsadowy ma wyprowadzać na ekran określone meldunki, niezależnie od ewentualnie wyłączonego echa, meldunki te należy umieścić bezpośrednio w zleceniu **ECHO**. W tym przypadku zlecenie **ECHO** działa podobnie jak **REM**, lecz niezależnie od stanu przełącznika echa. Uwaga! W komentarzu nie wolno używać nawiasów kątowych '<' ani '>', stosowanych do przełączania strumieni! Odnosi się to także do instrukcji **REM** i **PAUSE**.

REM komunikat lub **REMARK komunikat**

Instrukcja: **REM** służy do wyprowadzania na ekran dowolnych meldunków, informujących np. o stanie zaawansowania realizacji zadania, albo do komentowania zawartości samego programu wsadowego. Komunikat musi się mieścić w jednym wierszu, może też być pusty. Jeżeli zajdzie potrzeba wyprowadzania dłuższych meldunków lub objaśnień, można posłużyć się kilkoma

kolejnymi instrukcjami **REM**, albo też zapisać treść komunikatu w oddzielnym pliku dyskowym, wyprowadzanym następnie na ekran pojedynczym zleceniem **TYPE**, umieszczonym w pliku wsadowym. Wyprowadzanie komunikatów zawartych w pliku wsadowym w zleceniach **REM** i **REMARK** można zablokować umieszczając wcześniej w pliku tym zlecenie: **ECHO OFF**.

PAUSE komunikat

Instrukcja: PAUSE powoduje wstrzymanie realizacji następnych instrukcji pliku wsadowego do chwili naciśnięcia dowolnego klawisza. Po napotkaniu w pliku wsadowym instrukcji PAUSE, na ekranie pojawi się komunikat:

Strike any key when ready...

W celu kontynuacji programu naciśnij dowolny klawisz

Aby zwrócić uwagę operatora i udzielić mu niezbędnych wskazówek, można (choć nie trzeba) w instrukcji **PAUSE** umieścić dowolny komunikat. Nie będzie on jednak widoczny po wyłączeniu echa (ECHO OFF). Aby przerwać realizację programu wsadowego, należy użyć klawiszy **(Ctrl)** + **(C)** lub **(Ctrl)** + **(Break)**. Na ekranie pojawi się wówczas pytanie:

Terminate batch job (Y/N)?

Czy zakończyć zadanie wsadowe? (Klawisz (Y) potwierdza).

Instrukcja **PAUSE** umożliwia realizację programu wsadowego krok po kroku, pozostawiając za każdym razem użytkownikowi decyzję o jego kontynuacji bądź przedwczesnym przerwaniu (np. w zależności od uzyskanych wcześniej rezultatów). Przykład:

PAUSE Wstaw do stacji ⟨A⟩ dysk z danymi!

Na ekranie pojawi się komunikat wzywający do zmiany dyskietki, a program będzie kontynuowany dopiero po naciśnięciu dowolnego klawisza.

FOR %%zmienna IN (lista argumentów) DO zlecenie

Instrukcja: **FOR** umożliwia powtarzanie tego samego zlecenia dla pewnego zestawu argumentów i poza nazwą niewiele ma wspólnego z analogiczną instrukcją języka BASIC, Pascal, C czy CLIPPER. Nazwa zmiennej może być dowolnym znakiem, poza cyframi 0..9. W miejsce tej zmiennej, zapisanej w instrukcji, podczas przetwarzania wsadowego będą kolejno podstawiane parametry z listy argumentów. W przypadku, gdy któryś z argumentów jest szablonem nazwy pliku, zawierającym znaki zastępcze (np. TEST.∗ lub PROG∗.PAS), to w instrukcji zostaną podstawione kolejno wszystkie pliki katalogu roboczego, zgodne z tym szkieletem. Instrukcji FOR w pliku wsadowym nie można zagnieżdżać (instrukcja FOR nie może wystąpić po DO jako zlecenie wewnętrzne w innej instrukcji FOR). Przykład:

```
FOR %%x IN (RAPORT.TXT MELDUNEK.TXT NOT.TXT) DO TYPE %%x
```

W powyższej instrukcji zmienna %%x przyjmie kolejno nazwy plików, wyspecyfikowanych na liście, w wyniku czego na ekran konsoli zostanie wyprowadzona zawartość wszystkich trzech plików. Inny przykład:

```
FOR %%a IN (*.ASM, *.PAS, *.BAS) DO COPY A:%%a C:
```

Instrukcja ta spowoduje skopiowanie na dysk ⟨C⟩ wszystkich plików z rozszerzeniami: .ASM, .PAS i .BAS, zawartych w katalogu roboczym dysku ⟨A⟩:

IF warunek zlecenie lub **IF warunek instrukcja wsadowa**

Instrukcja: **IF** służy do warunkowej realizacji określonych zleceń. Zlecenie tzw. wewnętrzne, zawarte w instrukcji IF i umieszczone po warunku, zostanie wykonane tylko wtedy, jeśli podany warunek jest spełniony (prawdziwy). Warunek ten może przyjąć jedną z wymienionych niżej postaci:

ERRORLEVEL liczba

Warunek jest spełniony wtedy, gdy ostatnio zrealizowany program przekazał kod zakończenia nie mniejszy, niż wynosi wartość argumentu **ERRORLEVEL**. Kod zakończenia jest przekazywany systemowi operacyjnemu przez kończący pracę program, a jego interpretacja może być różna. Trzeba spojrzeć do dokumentacji programu. Przyjęła się konwencja, że kod zerowy oznacza zakończenie bezbłędne, a kody większe — zakończenie z powodu błędu, przy czym poważniejszym błędom odpowiadają wyższe kody. Np. w TURBO-Pascalu do przekazywania systemowi operacyjnemu kodu zakończenia służy procedura **Halt**.

tekst 1 = = tekst 2

Powyższy warunek jest spełniony, jeżeli po ewentualnym wcześniejszym podstawieniu parametrów aktualnych obydwa teksty są identyczne. Żaden z tekstów nie może zawierać dodatkowych separatorów, np. spacji. Oto prościutki przykład: program HASLO.BAT będzie kontynuowany tylko po podaniu w linii zlecenia właściwego hasła, tzn. po zleceniu: HASLO nowicjusz.

```
ECHO OFF
      IF %1 ==nowicjusz GOTO DALEJ
      GOTO KONIEC
:DALEJ
........     instrukcje
:KONIEC
```

EXIST Dysk: ścieżka\nazwa pliku.rozszerzenie

Warunek jest spełniony, jeśli plik o podanej specyfikacji został odnaleziony.

Przed każdym warunkiem można umieścić operator negacji logicznej **NOT**, odwracający warunek, np. z dwóch poniżej podanych instrukcji warunkowych wykonana zostanie tylko jedna (makroasembler MASM podaje kod zakończenia 1 i w razie wykrycia błędów w asemblowanym programie):

```
MASM %1,,,;
IF ERRORLEVEL 1 ECHO Wykryto blad - Koniec pracy
IF NOT ERRORLEVEL 1 ECHO Praca jest kontynuowana
```

Oto inny przykład:

```
IF EXIST PROGRAM1.BAS GWBASIC PROGRAM1
```

Jeżeli plik: PROGRAM1.BAS znajduje się w roboczym katalogu, to zostanie wywołany interpreter języka GWBASIC z tym plikiem w charakterze parametru.

GOTO etykieta

Instrukcja: **GOTO** pozwala na tworzenie w plikach wsadowych typu **.BAT** pętli, obejmujących wiele zleceń. Powoduje ona skok bezwarunkowy do linii pliku wsado-

wego następującej po wymienionej etykiecie. Aby uzyskać instrukcję skoku warunkowego, należy użyć instrukcji **GOTO** w instrukcji warunkowej **IF**. Oto przykład:

```
:PETLA
    TYPE KOMENT.TXT
    REM TO JEST PRZYKLAD PETLI W PRZETWARZANIU WSADOWYM
    GOTO PETLA
```

Powyższy plik wsadowy będzie cyklicznie listować na konsolę zawartości pliku tekstowego: KOMENT.TXT, na przemian z napisem: **TO JEST PRZYKLAD PETLI W PRZETWARZANIU WSADOWYM**. Warto zapamiętać, że w pliku wsadowym etykieta jest poprzedzana znakiem dwukropka (:) i musi występować w linii samotnie.

SHIFT

Instrukcja: **SHIFT** pozwala użyć w pliku wsadowym większej liczby parametrów rzeczywistych niż 10 oraz rozwiązać problem zmiennej liczby parametrów. Normalnie liczba parametrów rzeczywistych jest ograniczona liczbą zmiennych: %0..%9. Pierwszych dziesięć parametrów aktualnych jest przypisywanych zmiennym %0 do %9, dla reszty brak zmiennych roboczych. Wobec tego zlecenie SHIFT „przesuwa" zbiór parametrów aktualnych o jedną pozycję w lewo: pierwszy parametr rzeczywisty jest odrzucany, drugi zajmuje miejsce pierwszego, trzeci — miejsce drugiego, ..., jedenasty wchodzi na miejsce dziesiątego i jest przypisywany zmiennej 9. Używając wielokrotnie zlecenia **SHIFT**, można udostępnić kolejne parametry. Przykład: Plik ZADANIE.BAT wywołano z konsoli w sposób następujący:

```
A> ZADANIE NOTATKA KOMENT TEKST DANE
```

Przypuśćmy, że plik ZADANIE.BAT zawierał następujący program przetwarzania wsadowego:

```
TYPE %1.TXT
SHIFT
TYPE %1.TXT
SHIFT
TYPE %1.TXT
TYPE %2.TXT
```

W efekcie zostaną kolejno wylistowane cztery pliki: NOTATKA.TXT, KOMENT.TXT, itd. Poniższy program wsadowy umożliwia natomiast wylistowanie dowolnej liczby plików z rozszerzeniem .TXT:

```
ECHO OFF
:PETLA
    TYPE %1.TXT
    SHIFT
    IF EXIST %1.TXT GOTO PETLA
```

Ten sam program wsadowy można było oczywiście zapisać inaczej, np. tak:

```
ECHO OFF
:PETLA
    SHIFT
    IF NOT EXIST %0.TXT GOTO KONIEC
    TYPE %0.LST
    GOTO PETLA
:KONIEC
```

Na zakończenie przedstawimy dwa przykłady kompletnych programów wsadowych, przeznaczonych do automatycznej instalacji oprogramowania. Pierwszy ma zainstalować na dysku sztywnym wyimaginowany system KAKTUS, dostarczony na dyskietkach. W tym celu program zakłada na dysku sztywnym podkatalog KAKTUS, po czym przystępuje do kopiowania, za każdym razem prosząc użytkownika o wsunięcie właściwej dyskietki. Jak skontrolować, czy użytkownik wsunął właściwy dysk? Najprościej zbadać, czy na dysku tym znajduje się określony plik. Przyjmujemy, że na dyskietce KAKTUS__A występuje m.in. plik KAKTUS1.COM, a na dyskietce KAKTUS__B — plik KAKTUS.OVL. Program wsadowy będzie powtarzał prośbę o wsunięcie właściwej dyskietki aż do skutku:

```
ECHO OFF
C:
MD\KAKTUS
CD\KAKTUS
:PETLA1
    ECHO Wsun do stacji [A] dyskietke KAKTUS_A
    PAUSE
    IF NOT EXIST A:\KAKTUS1.COM GOTO PETLA1
COPY A:\*.*
:PETLA2
    ECHO Wsun do stacji [A] dyskietke KAKTUS_B
    PAUSE
    IF NOT EXIST A:\KAKTUS.OVL GOTO PETLA2
COPY A:\*.*
```

Oto przykład bardziej złożony pliku wsadowego, służącego do instalacji polskojęzycznego pakietu adaptacyjnego POLONUS do dBASE III Plus. Istota instalacji polega na skopiowaniu jednego z dwóch plików: POLONUS.CGA lub POLONUS.HGC do pliku POLONUS.COM. Należy przy tym wykryć ewentualne błędy i udzielić użytkownikowi wskazówek:

```
ECHO OFF
REM ** Roland Wacławek, Siemianowice Sl. 11 stycznia 1988 **
CLS
ECHO Program zainstaluje pakiet POLONUS  dla karty graficznej
ECHO CGA albo dla monochromatycznej karty graficznej HERCULES
FOR %%x IN (HGC hgc CGA cga) DO IF [%1]==[%%x] GOTO POPRAWNE
ECHO  Musisz podac zlecenie:  INSTALUJ HGC dla karty Hercules
ECHO  albo zlecenie:  INSTALUJ CGA  dla karty grafiki barwnej
GOTO KONIEC
:POPRAWNE
IF EXIST exist polonus.%1 GOTO jest
ECHO Na dysku brak kompletnego zestawu plikow instalacyjnych!
GOTO KONIEC
:JEST
GOTO KARTA%1
:KARTACGA
COPY polonus.cga polonus.com >nul
ECHO Pakiet POLONUS zainstalowany dla karty grafiki barwnej
GOTO KONIEC
:KARTAHGC
COPY polonus.hgc polonus.com >nul
ECHO Pakiet POLONUS zainstalowany dla karty Hercules
:KONIEC
```

Instrukcja **FOR** służy tutaj do sprawdzenia, czy parametr istnieje i czy jest jednym z dopuszczalnych słów: CGA lub HGC (należy przewidzieć przypadek wypisania parametru małymi lub dużymi literami). Klauzula: **>nul**, zawarta w zleceniu **COPY**, przełącza strumień danych, kierowanych pierwotnie do monitora, do urządzenia pozornego (zerowego) **NUL**. Sprawia to, że na ekranie nie ukażą się komunikaty wyprowadzane przez zlecenie. Sztuczkę tę można zresztą stosować także w innych zleceniach PC-DOS, kiedy trzeba stłumić wyprowadzane przez nie meldunki.



ANDRZEJ M. GROSSMAN

PROCEDURY ZE ZMIENNYMI LOKALNYMI W BASIC ZX SPECTRUM

O tym, że możliwość korzystania z procedur ze zmiennymi lokalnymi znacznie ułatwia pisanie programów, nie trzeba chyba nikogo przekonywać. Procedury są dostępne w wielu językach wysokiego poziomu, a ostatnio, wiele dialektów języka BASIC wzbogaciło się o to pożyteczne narzędzie. Nawet poczciwe Spectrum dysponuje procedurami w dialektach Beta-BASIC 3.0 i Mega-BASIC, które, poza szeregiem zalet, mają niestety podstawową wadę — zajmują pokaźną część dostępnej dla programisty pamięci komputera.

Chcąc adaptować długie programy z języka Fortran musiałem sobie jakoś poradzić i tak powstał prezentowany tu PROCLOK. Od prawie dwóch lat pracuję z jego nieco uproszczoną wersją i na ogół spełnia dobrze swoje zadania. A że Spectrum nie nadaje się do tzw. poważnych zastosowań? Możliwe, ale instytucji w której pracuję jakoś nie stać było przez długi czas na kupno IBM-a i podejrzewam, że nie jest to przypadek odosobniony.

Oczywiście, jeżeli piszecie niezbyt długi program i wystarczy Wam mniejsza dokładność obliczeń (czterobajtowa reprezentacja liczb), zdecydowanie polecam Pascal, a PROCLOK niech będzie po prostu okazją do pogrzebania w procedurach systemowych.

PROCLOK zajmuje 605 bajtów od adresu 64763 do początku UDG i nie jest relokowalny. Korzystanie z niego ułatwią, mam nadzieję, załączone przykłady. Poniżej podaję nieco szczegółów.

Przed wczytaniem z taśmy wygenerowanego przez loader kodu maszynowego trzeba wykonać CLEAR 64762.

Program w języku BASIC przed pierwszym wywołaniem procedury musi, przy każdym uruchomieniu, wykonać RANDOMIZE USR 64763. Sposób deklaracji procedury

nr linii REM PROC nazwa
nr linii REM [lista1; lista2]
......
TREŚĆ PROCEDURY
......
nr linii RETURN

Procedury lepiej umieścić na początku programu, gdyż szybciej są znajdowane przez PROCLOK.

Wywoływanie procedury

. . GO SUB PROC nazwa: REM [lista3; lista4]
(w jednej linii)

Słowo zastrzeżone PROC — dużymi literami. Nazwa o dowolnej długości dużymi lub małymi literami — w tym przypadku są rozróżniane! Pierwszy znak nie może być cyfrą.

Listy parametrów są opcjonalne. Wartości parametrów umieszczonych za średnikiem są przekazywane z powrotem do programu wywołującego procedurę. W przypadku braku list 1 i 3, zaraz po nawiasie należy umieścić średnik. Jest również dopuszczalny zapis [lista2; lista2] oraz [lista4; lista4], ale niepotrzebnie wydłuża to deklarację i opłaca się tylko w przypadku szybkiego przepisywania zmiennych, o czym dalej.

W listach parametrów mogą występować tylko zmienne. Nie można używać liczb, wyrażeń i funkcji. Tablice liczbowe i alfanumeryczne oraz proste zmienne alfanumeryczne przekazywane są tylko w całości.

Tabela 1 Wpływ struktury list na wartości zmiennych

lista przy wywołaniu	lista w procedurze	efekt w procedurze	efekt po powrocie
[A,A,X]	[M,N,U]	M=A,N=A,U=X	A=A,X=X
[A;A]	[M;N]	M=A,N=A	A=N
[A,B;C,D]	[K,M;M,M]	K=A,M=B (pierwsze przypisanie)	A=A,B=B,C=M D=M
[A;A,A]	[K;M,N]	K=A,M=A,N=A	A=N (ostatnie przypisanie)

Długość przekazywanych prostych zmiennych alfanumerycznych musi być równa długości zmiennych przyjmujących wartości. (Procedura nie może zmieniać „wymiarów" zmiennych). W listach parametrów w procedurze można stosować tylko nazwy jednoliterowe.

Zasady nazewnictwa zmiennych prostych jak w BASIC ZX Spectrum, natomiast po nazwach zmiennych tablicowych stosuje się puste nawiasy bez indeksów np. a(), B$().

Zmienne występujące w liście parametrów przy wywołaniu procedury, muszą na danym poziomie być dostępne (deklaracja LET, DIM FOR. . lub przekazanie z poprzedniego poziomu).

Nie istnieją zmienne globalne, dostępne z procedury. Parametry można przekazywać tylko z i do procedur mających bezpośrednio sąsiadujące obszary zmiennych. Zmian zmiennych na innych poziomach można dokonywać tylko przekazując zmienne kolejno na poziomy pośrednie, co niestety dodatkowo zajmuje pamięć, zwłaszcza w przypadku tablic.

Zmienne typu FOR...NEXT przekazywane są jako zmienne proste do procedury i mogą przyjmować wartości zmiennych prostych z procedury. Nie można natomiast przekazywać zmiennych typu FOR...NEXT z procedury.

W przypadku powtarzania się nazw zmiennych otrzymać można różne efekty, w zależności od tego, w którym miejscu list zmienne te są wyszczególnione. Przykłady zawiera tablica 1.

Z procedury można wyjść tylko przez RETURN (GO TO może wprowadzić straszny bałagan) lub zatrzymać program przez użycie STOP, czy BREAK, ale użycie CONTINUE nie uruchomi wtedy prawidłowo programu.

Szybkie przepisywanie zmiennych

W języku BASIC dla ZX Spectrum można przypisać np. zmiennej prostej A wartość zmiennej B stosując LET A = B. PROCLOK umożliwia podobną operację również dla całych tablic. W tym celu trzeba np. wykonać GO SUB PROC PRZEPISZ:REM [A(); B()], a procedura może mieć postać:

```
REM PROC PRZEPISZ
REM [A(); A()]
RETURN
```

Oczywiście jedna procedura może przepisać więcej tablic zarówno liczbowych, jak i alfanumerycznych.

Sygnalizacja błędów

Zastosowałem bardzo uproszczoną analizę składni, w związku z czym interpreter nie sygnalizuje następujących błędów:

Rys. 1 Organizacja obszaru zmiennych

```
          :                          :
          :          BASIC           :
VARSO --> :--------------------------:<---,
          :         Zmienne          :    :
          :   poziomu podstawowego   :    :
          :--------------------------:    :
          :           128            :    :
          :--------------------------:    :
          :     Nr linii z lista     :    :
          :   parametrów  PROC  1    :    :
          :--------------------------:    :
          :     Wartość (VARS)       :____:
          :   poziomu podstawowego   :
          :--------------------------:<---,
          :         Zmienne          :    :
          :         PROC  1          :    :
          :--------------------------:    :
          :           128            :    :
          :--------------------------:    :
          :     Nr linii z lista     :    :
          :   parametrów  PROC  2    :    :
          :--------------------------:    :
          :     Wartość (VARS)       :____:
          :         PROC 1           :
          :--------------------------:
          :                          :
          ~          itd.            ~
          :                          :
          :--------------------------:<---,
          :         Zmienne          :    :
          :        PROC  n-1         :    :
          :--------------------------:    :
          :           128            :    :
          :--------------------------:    :
          :     Nr linii z lista     :    :
          :   parametrów  PROC  n    :    :
          :--------------------------:    :
          :     Wartość (VARS)       :____:
          :        PROC  n-1         :
VARST-->  :--------------------------:<--VARS
          :         Zmienne          :
          :         PROC n           :
          :--------------------------:
          :           128            :
          :--------------------------:<--ELINE
          :      BUFOR  EDYTORA      :
```

```
30 **********************************
40 ;        PROCLOK 1.1
50 ;     A.M GROSSMAN 1988
60 **********************************
70             ORG    64763
90
100 ***  INICJALIZACJA ***
110 INIC       LD     HL,(23627)
120            LD     (VARSO).HL
130            LD     HL,(23613)
140            LD     DE.OBSLBL
150            LD     (HL),E
160            INC    HL
170            LD     (HL).D
180            RET
190
200 ***** PROGRAM GLOWNY *****
210 OBSLBL            LD (IY+71).0
220            DEC    SP
230            DEC    SP
240            LD     A,(23610)
250            CP     22
260            JR     NZ,BVAR
270            LD     HL,(23618)
280            LD     BC,10000
290            SBC    HL,BC
300            JR     NZ,KOMBLD
310            LD     (IY+71),128
320            CALL   PRZEKZ
330            CALL   LIKW
340            CALL   #1F23
350            JR     BAS
360 BVAR       CP     1
370            JR     NZ,KOMBLD
380            LD     HL,(23645)
390 P1         LD     A,(HL)
400            DEC    HL
410            CP     165: RND
420            JR     C,P1
430            CP     237; GO SUB
440            JR     NZ,KOMBLD
450            INC    HL
460            CALL   LPROC
470            JR     NZ.KOMBLD
480            CALL   #2D2B
490            CALL   #1EED
500            CALL   TWORZ
510            CALL   PRZEKZ
520            LD     H.1
530            EX     (SP),HL
540            INC    SP
550            LD     BC.10000
560            PUSH   BC
570            PUSH   HL
580            LD     (23613),SP
590            CALL   #1F05 - 3
600 BAS        LD     (IY+0),#FF
610            SET    7,(IY+1)
620            JP     #1B7D
630 KOMBLD            CALL LIKW
640            JR     C,KOMBLD
650            INC    SP
660            INC    SP
670            JP     #1303
680 **********************************
690 * P O D P R O G R A M Y *
700
710 * SZUKANIE PROCEDURY *
720 LPROC      INC    HL
730            LD     (23629),HL
740            LD     DE,WZOR
750            CALL   PORWN
760            RET    NZ
770            LD     HL,(23635)
780 LINREM            LD E.234 ;REM
790            CALL   #1D8B+1
800            JP     C,#1BEC
810            INC    HL
820            LD     DE,(23629)
830            CALL   PORWN
840            JR     NZ,NASTP
850            LD     A,13; ENTER
860            CP     (HL)
870 NASTP      EX     AF,AF'
880            LD     HL,(23618)
890            INC    HL
900            CALL   #196E
910            EX     AF,AF'
920            JR     NZ,LINREM
930            LD     B,(HL)
940            INC    HL
950            LD     C,(HL)
960            RET
970
980 *POROWNANIE NAZW*
990 PORWN      LD     A,(DE)
1000           CP     ":"
1010           RET    Z
1020           CP     13; ENTER
1030           RET    Z
1040           CP     (HL)
1050           RET    NZ
1060           INC    HL
1070           INC    DE
1080           JR     PORWN
1090
1100 *PRZEPISYWANIE ZMIENNYCH*
1110 PRZEKZ           LD HL,(23627)
1120           DEC    HL
1130           DEC    HL
1140           DEC    HL
1150           LD     D,(HL)
1160           DEC    HL
1170           LD     E,(HL)
1180           EX     DE,HL
1190           CALL   #196E
1200           CALL   LISTA
1210           RET    NZ
1220           INC    (IY+10)
1230           INC    (IY+13)
1240 ZMNL      CALL   ZMNNUM
1250           LD     HL,(POCZML)
1260           LD     (23645),HL
1270           RES    0,(IY+71)
1280           CALL   #28B2
1290           PUSH   AF ;CF
1300           LD     (23629),HL
1310           CALL   SKLD
1320           LD     (POCZML),HL
1330           LD     B,A
1340           PUSH   BC
1350           CALL   PPOZ
1360           CALL   ZMNNUM
1370           LD     HL,(POCZMP)
1380           LD     (23645),HL
1390           CALL   #28B2
1400           JP     C,#1C2E
1410           PUSH   HL
1420           BIT    5,C
1430           JR     Z,BEZZM
1440           SET    6,C
1450 BEZZM     CALL   SKLD
1460           LD     (POCZMP),HL
1470           CALL   LPOZ
1480           POP    HL
1490           POP    DE;E
1500           XOR    D
1510           JP     NZ,BLADE
1520           LD     A,C
1530           XOR    E
1540           AND    %11100000
1550           JR     NZ,BLADQ
1560 PRZEP     POP    AF
1570           JR     C,PRZEP0
1580           LD     A,D
1590           BIT    7,(IY+71)
1600           JR     Z,KNCLST
1610 PRZEP0           PUSH DE
1620           BIT    5,C
1630           LD     BC,6
1640           CALL   Z,#19B8
1650           POP    DE
1660           PUSH   BC
1670           PUSH   HL
1680           PUSH   DE
1690           BIT    7,(IY+71)
1700           JR     Z,PRZEP1
1710           PUSH   BC
1720           LD     HL,(23629)
1730           CALL   #19B8
1740           EX     (SP).HL
1750           XOR    A
1760           SBC    HL,BC
1770           JR     NZ,BLADQ
1780           POP    HL
1790           POP    DE
1800           JR     PRZEP2
1810 PRZEP1           LD HL,(23641)
1820           DEC    HL
1830           CALL   #1655
1840           INC    HL
1850           POP    DE
1860           LD     (HL).E
1870           EX     (SP),HL
1880 PRZEP2           LD A.D
1890           POP    DE
1900           POP    BC
1910           INC    HL
1920           INC    DE
1930           DEC    BC
1940           LDIR
1950 KNCLST           CP "I"
1960           JP     NZ ZMNL
1970 KONIEC           RET
1980 BLADQ     RST    8
1990           DEFB   25
2000
2010 :*SZUKANIE POCZATKOW LIST
2015 : PARAMETROW               *
2020 LISTA     INC    HL
2030           INC    HL
2040           INC    HL
2050           CALL   CZYREM
2060           PUSH   AF
2070           LD     (POCZML).HL
2080           LD     HL.(23613)
2090           INC    HL
2100           INC    HL
2110           LD     E.(HL)
2120           INC    HL
2130           LD     D.(HL)
2140           INC    HL
2150           PUSH   HL
2160           EX     DE.HL
2170           CALL   #196E
2180           POP    BC
2190           LD     E.0
2200           LD     A.(BC)
2210           LD     D.A
2220           CALL   #198B-3
2230           CALL   CZYREM
2240           LD     (POCZMP).HL
2250           JR     NZ.LISTA1
2260           POP    AF
2270           RET    Z
2280           JR     BLADE
2290 LISTA1           POP AF
2300           RET    NZ
2310 BLADE     RST    8
2320           DEFB   #0D
2330
2340 *SZUKANIE "REM I" i ":"*
2350 CZYREM           INC HL
2360           LD     DE.WZOR2
2370           CALL   PORWN
2380           RET    NZ
2390           LD     A.(HL)
2400           CP     ":"
2410           JR     Z.SREDN1
2420           XOR    A
2430           BIT    7.(IY+71)
2440           RET    Z
2450 SREDN     LD     A.(HL)
2460           CP     ":"
2470 SREDN1           INC HL
2480           RET    Z
2490           CP     "I"
2500           JR     NZ.SREDN
2510           AND    A :Z=0
2520           RET
2530
2540 *ZAMIANA PPC i NEWPPC.
2545 : SUBPPC i NSPPC         *
2550 ZMNNUM           LD B.3
2560           LD     HL.23621
2570           LD     DE.23618
2580 ZMNNU1           LD C.(HL)
2590           LD     A.(DE)
2600           LD     (HL).A
2610           LD     A.C
2620           LD     (DE).A
2630           INC    HL
2640           INC    DE
2650           DJNZ   ZMNNU1
2660           RET
2670
```

```
2680 *KONTROLA SKLADNI I UZUP
2685 ;BITU 7 TYPU ZMIENNEJ     *L2600
2690 SKLD     RST   #18
2700          CP    "("
2710          JR    Z.SET
2720          CP    "$"
2730          JR    NZ.SKLD2
2740          RST   #20
2750          CP    "("
2760          JR    NZ,SKLD1
2770 SET      SET   7.C
2780          RST   #20
2790          CP    ")"
2800          JR    NZ,BLADSK
2810 SKLD1    RST   #20
2820 SKLD2    INC   HL
2830          CP    ","
2840          RET   Z
2850          CP    ";"
2860          RET   Z
2870          CP    "]"
2880          RET   Z
2890 BLADSK         RST 8
2900          DEFB  #0B
2910
2920 *LOKALNY OBSZAR ZMIENN.*
2930 TWORZ    LD    HL.(23641)
2940          DEC   HL
2950          LD    BC.5
2960          CALL  #1655
2970          INC   HL
2980          LD    (HL).128
2990          INC   DE
3000          PUSH  DE
3010          INC   HL
3020          LD    DE.(23618)
3030          LD    (HL).E
3040          INC   HL
3050          LD    (HL).D
3060          INC   HL
3070          LD    DE,(23627)
3080          LD    (HL),E
3090          INC   HL
3100          LD    (HL).D
3110          POP   HL
3120          LD    (23627).HL
3130          RET
3140
3150 *LIKWIDACJA ZMIENN.LOK.*
3160 LIKW     BIT   0.(IY+71)
3170          CALL  NZ.LPOZ
3180 LIKW1    LD    DE,(VARS0)
3190          LD    HL,(23627)
3200          XOR   A
3210          SBC   HL.DE
3220          RET   Z
3230          CALL  PPOZ1
3240          DEC   HL
3250          DEC   HL
3260          LD    DE,(23641)
3270          EX    DE.HL
3280          CALL  #19E5
3290          SCF
3300          RET
3310
3320 *POPRZEDNI POZIOM*
3330 PPOZ     SET   0.(IY+71)
3340 PPOZ1    LD    HL.(23627)
3350          LD    (VARST).HL
3360          DEC   HL
3370          LD    D.(HL)
3380          DEC   HL
3390          LD    E.(HL)
3400          LD    (23627).DE
3410          RET
3420
3430 *POWROT DO POZ.LOKALNEGO*
3440 LPOZ     LD    HL.(VARST)
3450          LD    (23627).HL
3460          RES   0.(IY+71)
3470          RET
3480
3490 ***PAMIEC PROGRAMU***
3500 VARS0    DEFW  0
3510 VARST    DEFW  0
3520 POCZML         DEFW 0
3530 POCZMP         DEFW 0
3540
3550 ***WZORY DLA PORWN***
3560 WZOR     DEFM  "PROC :"
3570 WZOR2    DEFB  234,"[",";"
```

Nie ma informacji o powtórzeniu się nazw procedury. Pod uwagę brana jest tylko pierwsza.

W przypadku wystąpienia więcej niż jednego średnika w listach parametrów też uwzględniany jest tylko pierwszy.

Sygnalizowane są następujące błędy:

2 *Variable not found* — dla listy parametrów przy wywołaniu procedury
— w programie wywołującym nie występuje taka zmienna

jw — dla treści procedury
— nie przekazano zmiennej do procedury

4 *Out of memory* — zmienne nie mieszczą się w pamięci

C *Nonsense in BASIC* — błąd składni w liście parametrów

E *Out of data* — dla listy parametrów przy wywołaniu procedury
— jedna z list parametrów jest za krótka lub nie istnieje
— brak nawiasów [lub] w jednej z list
— średnik w listach na różnych pozycjach

jw. — dla RETURN
— brak średnika na początku jednej z list

N *Statement lost* — dla linii z wywołaniem procedury
— brak procedury o takiej nazwie
— po deklaracji nazwy procedury występują w tej samej linii dalsze instrukcje lub lista parametrów

Q *Parameter error* — dla linii z wywołaniem procedury
— niezgodność typów zmiennych w listach
— wieloliterowa nazwa w liście parametrów w procedurze

jw. — dla RETURN
— procedura zmienia „wymiary" łańcucha lub tablicy, albo usiłuje przekazać zmienną typu FOR...NEXT

UWAGA!

Program zmienia adres procedury obsługi błędu i będzie kolidował z innymi programami działającymi na tej samej zasadzie. W przypadku wystąpienia komunikatu o błędzie (także: O.K., STOP, BREAK), likwidowane są wszystkie obszary zmiennych lokalnych oraz przywracany jest standardowy adres procedury obsługi błędu. W trakcie pracy program wykorzystuje komórkę pamięci o adresie 23681.

Zasada działania programu

Organizacja obszaru zmiennych podczas korzystania z procedur przedstawiona jest na załączonym rysunku. Jest to, jak widać, wielokrotne powielanie organizacji obszaru zmiennych interpretera języka BASIC ZX Spectrum, z tą niewielką różnicą, że po znaczniku końca obszaru zmiennych (128) wykorzystano cztery bajty na pamięć pomocniczą lokalną, która zawiera numer linii z listą parametrów w procedurze i adres początku obszaru zmiennych poprzedniego poziomu. Konstrukcja taka nie limituje głębokości zagnieżdżeń w procedury, pozwala na łatwy powrót do poprzedniego poziomu, a interpreter wykonuje procedurę jak normalny program. Ingerencja potrzebna jest tylko w momencie wywołania oraz powrotu z procedury i w tym celu został wykorzystany mechanizm obsługi błędu.

Szczupłe ramy „InforMika" zmusiły mnie do usunięcia komentarzy w listingu programu źródłowego i zamieszczenia jedynie krótkich objaśnień w tekście. Bliższe dane na temat wielu procedur i zmiennych systemowych wykorzystywanych przez PROCLOK można znaleźć w następujących pozycjach:

„Przewodnik po ZX Spectrum" — K. Kuryłowicz, D. Madej i K. Marasek;
„Tajniki ZX Spectrum" — A. Kadlof;
„The Complete Spectrum ROM Disassembly" — I. Logan, F. O'Hara.

Uwagi do listingu — w nawiasach podano numery linii

INICJALIZACJA (100—160) — zapamiętuje adres początku obszaru zmiennych ZX Spectrum w zmiennej VARSØ i zmienia wartość zmiennej systemowej ERR__SP, czyli adres procedury obsługi błędów na adres OBSLBL

OBSLBL (210—670) — główny program obsługi błędów uruchamiany w przypadku wystąpienia błędu w trakcie wykonywania programu w ZX BASIC. Kod błędu pobierany jest ze zmiennej systemowej ERR__NR

(240). Jeżeli jest różny od 22 lub 1, następuje skok do linii 630, wywoływana jest procedura likwidująca ewentualne obszary zmiennych lokalnych i wykonywany jest skok do systemowej procedury obsługi błędów (670).

Interpreter języka BASIC ZX Spectrum wywołanie procedury: GO SUB PROC nazwa traktuje, jak odwołanie się do nie istniejącej zmiennej PROC nazwa i sygnalizuje błąd 1 — *Variable not found.* Procedura sprawdza (380—440) czy przed nazwą zmiennej wystąpiło GO SUB. Jeżeli tak, wywoływana jest procedura szukająca w liniach programu w języku BASIC żądanej deklaracji (460), numery linii i instrukcji, do których ma nastąpić powrót z procedury, przekazywane są na stos GO SUB (480—490). Następnie tworzony jest obszar zmiennych lokalnych (500) i zmienne zostają przekazane do tego obszaru (510).

Na stos GO SUB przekazywany jest numer nie istniejącej linii 10 000 oraz instrukcji 1 (520—570) i następuje powrót do interpretera języka BASIC w celu wykonania żądanej procedury. Po napotkaniu instrukcji RETURN interpreter pobiera ze stosu GO SUB nr linii 10 000 i sygnalizuje błąd 22 — *Statement lost*, gdyż numery linii w programie mogą przyjmować wartości do 9999.

Procedura obsługi błędów sprawdza, czy błąd nr 22 został spowodowany właśnie taką sytuacją i jeżeli tak, to do programu wywołującego przekazywane są wartości zmiennych lokalnych (250—300), a ze stosu GO SUB pobierane są właściwe numery linii oraz instrukcji, do których ma nastąpić powrót (340) i przekazywane do interpretera.

LPROC (720—950) — sprawdza czy po GO SUB użyto słowa zastrzeżonego PROC, jeżeli nie, następuje powrót z wyzerowanym wskaźnikiem Z. W przeciwnym przypadku wyszukiwane są kolejne linie z REM (760—800) i tekst za REM porównywany jest z nazwą wywołanej procedury (830—840, 870—920). Po znalezieniu ustawiany jest wskaźnik Z, a rejestry BC zawierają nr linii z listą parametrów procedury, jeżeli nie znaleziono procedury — komunikat *Statement lost* (800).

PORWN (990—1080) — porównuje łańcuchy znaków o adresach w DE i HL. Koniec łańcucha oznaczają kody 13 lub 58. W przypadku zgodności łańcuchów wskaźnik Z = 1.

PRZEKAZ (1110—1990) — sprawdza poprawność składni list parametrów, zgodność typów zmiennych, tworzy i likwiduje obszary zmiennych lokalnych, przekazuje zmienne między obszarami. Bit 7, (IY + 71) pełni rolę wskaźnika. IY zawsze = 23610. W przypadku wyzerowanego bitu 7 tworzony jest nowy obszar zmiennych i przekazywane są tam wartości zmiennych z programu wywołującego, jeżeli bit 7 = 1, zwracane są z procedury wartości zmiennych umieszczone na liście za średnikiem.

LISTA (2020—2320) — wykorzystywana przez PRZEKAZ do lokalizacji list parametrów w wywoływanej procedurze i w instrukcji wywołania. Wskaźnik Z = 1 świadczy o braku obu list, przy braku jednej — komunikat *Out of DATA*.

CZYREM (2350—2520) — wykorzystywana przez procedurę LISTA do ustawienia HL na adres pierwszego znaku nazwy pierwszej zmiennej w liście, kiedy bit 7,

```
  10 REM ******************************
  20 REM *      PROCLOK 1.1         *
  30 REM *  A.M.Grossman 1988       *
  40 REM *        loader            *
  50 REM ******************************
  60 CLEAR 64762: LET A=64763: FO
R I=1 TO 25
  70 READ L$: LET L=LEN L$: LET S
=0: LET K=2
  80 LET A$=L$(K-1): LET B$=L$(K)

  90 LET C=(CODE A$-48-(7*(A$>":"
)))*16+CODE B$-48-(7*(B$>":"))
 100 IF K<L THEN  POKE A,C: LET S
=S+C: LET K=K+2: LET A=A+1: GO TO
 80
 110 IF S-256*INT (S/256)<>C THEN
  PRINT "BLAD W LINII ";I: STOP
 120 NEXT I
 130 DATA "2A4B5C2247FF2A3D5C110B
FD732372C9FD3647003B3B3A3A5CA6"
 140 DATA "FE1620192A425C011027ED
42204DFD364780CDB8FDCD08FFCD06"
 150 DATA "231F1833FE01203A2A5D5C
7E2BFEA538FAFEED202D23CD79FDE5"
 160 DATA "2027CD2B2DCDED1ECDE4FE
CDB8FD2601E333011027C5E5ED73F4"
 170 DATA "3D5CCD021FFD3600FFFDCB
01FEC37D1BCD08FF38FB3333C3030E"
 180 DATA "1323224D5C114FFFCDABFD
C02A535C1EEACD8C1DDAEC1B23EDDD"
 190 DATA "5B4D5CCDABFD20033E0DBE
082A425C23CD6E190820E046234EAB"
 200 DATA "C91AFE3AC8FE0DC8BEC023
1318F32A4B5C2B2B2B562B5EEBCD5E"
 210 DATA "6E19CD62FEC0FD340AFD34
0DCDB0FE2A4BFF225D5CFDCB47864C"
 220 DATA "CDB228F5224D5CCDC2FE22
4BFF47C5CD29FFCDB0FE2A4DFF2274"
 230 DATA "5D5CCDB228DA2E1CE5CB69
2802CBF1CDC2FE224DFFCD3CFFE167"
 240 DATA "D1AAC290FE79ABE6E02047
F138077AFDCB477E2837D5CB6901BC"
 250 DATA "0600CCB819D1C5E5D5FDCB
477E2811C52A4D5CCDB819E3AFED6E"
 260 DATA "42201DE1D1180B2A595C2B
CD551623D173E37AD1C123130BED1A"
 270 DATA "B0FE5DC2CFFDC9CF192323
23CD92FEF5224BFF2A3D5C23235ED8"
 280 DATA "235623E5EBCD6E19C11E00
0A57CD8819CD92FE224DFF2004F14E"
 290 DATA "C81802F1C0CF0D231155FF
CDABFDC07EFE3B2809AFFDCB477E50"
 300 DATA "C87EFE3B23C8FE5D20F7A7
C9060321455C11425C4E1A77791230"
 310 DATA "231310F7C9DFFE282809FE
24200DE7FE282007CBF9E7FE2920B1"
 320 DATA "0BE723FE2CC8FE3BC8FE5D
C8CF0B2A595C2B010500CD55162370"
 330 DATA "368013D523ED5B425C7323
7223ED5B4B5C732372E1224B5CC93C"
 340 DATA "FDCB4746C43CFFED5B47FF
2A4B5CAFED52C8CD2DFF2B2BED5B00"
 350 DATA "595CEBCDE51937C9FDCB47
C62A4B5C2249FF2B562B5EED534B10"
 360 DATA "5CC92A49FF224B5CFDCB47
86C90000000000000000050524F43F2"
 370 DATA "203AEA5B3AD9"
```

```
 10 REM ******************************
 20 REM *       PRZYKŁAD            *
 30 REM * MNOŻENIE  MACIERZY *
 40 REM ******************************
 50 REM Uaktywnienie PROCLOK
 60 RANDOMIZE USR 64763
 70 GO TO 510
 80 REM ****** PROCEDURY ******
 90 REM * WCZYTYWANIE WYMIARÓW*
 100 REM PROC WYMIARYMTX
 110 REM [:M,N,A$]
 120 INPUT "Nazwa macierzy :";A$
 130 PRINT "MACIERZ ";A$
 140 INPUT "Wymiary macierzy ";(A
$)'"Wierszy=";M,"Kolumn=";N
 150 PRINT "M=";M,"N=";N
 160 RETURN
 170
 180 REM *WCZYTYWANIE  MACIERZY*
 190 REM PROC WCZYTAJMTX
 200 REM [:M,N,A()]
 210 LET A$=" "
 220 GO SUB PROC WYMIARYMTX: REM
[;M,N,A$]
 230 PRINT "MACIERZ ";A$
 240 PRINT "Wprowdzaj dane wiersz
ami"
 250 FOR I=1 TO M: FOR J=1 TO N
 260 INPUT (A$;I;",";J);"=";A(I,J
)
 270 PRINT A(I,J),
 280 NEXT J: PRINT
 290 NEXT I
 300 RETURN
 310
 320 REM * MNOŻENIE MACIERZY A*B
 330 REM PROC MNOZENIEMTX
 340 REM [M,L,N,A(),B():C()]
 350 FOR K=1 TO L: FOR I=1 TO M
 360 FOR J=1 TO N
 370 LET C(I,J)=C(I,J)+A(I,K)*B(K
,J)
 380 NEXT J: NEXT I: NEXT K
 390 RETURN
 400
 410 REM *   WYDRUK MACIERZY   *
 420 REM PROC PISZMTX
 430 REM [A$,M,N,A()]
 440 PRINT "MACIERZ ";A$
 450 FOR I=1 TO M: FOR J=1 TO N
 460 PRINT A(I,J),
 470 NEXT J: PRINT
 480 NEXT I
 490 RETURN
 500
 510 REM *** PROGRAM  GŁÓWNY ***
 520 REM DEKLARACJA ZMIENNYCH PRO
GRAMU
 530 LET wierszy1=0: LET kolumn1=
0
 540 LET wierszy2=0: LET kolumn2=
0
 550 LET A$=" "
 560 INPUT "Maksymalny wymiar mac
ierzy :";MAX
 570 DIM A(MAX,MAX): DIM B(MAX,MA
X): DIM R(MAX,MAX)
 580
 590 REM WCZYTANIE MACIERZY AiB
 600 GO SUB PROC WCZYTAJMTX: REM
[;wierszy1,kolumn1,A()]
 610 GO SUB PROC WCZYTAJMTX: REM
[;wierszy2,kolumn2,B()]
 620 REM MACIERZ WYNIKOWA
 630 LET A$="R"
 640 GO SUB PROC MNOZENIEMTX: REM
[wierszy1,kolumn1,kolumn2,A(),B(
):R()]
 650 GO SUB PROC PISZMTX: REM [A$
,WIERSZY1,KOLUMN2,R()]
 660 REM KONIEC
```

```
 10 REM ******************************
 20 REM * PRZYKŁAD REKURENCJI *
 25 REM * RYSOWANIE KWADRATÓW *
 30 REM ******************************
 40 RANDOMIZE USR 64763
 50 REM *** PROGRAM GŁÓWNY ***
 60 LET X1=0: LET Y1=175: LET bo
k=175: LET skok=10
 70 GO SUB PROC KWADRATY: REM [s
kok,X1,Y1,bok]
 80 STOP
 90 REM ****** PROCEDURY ******
 100 REM PROC KWADRAT
 110 REM [X,Y,L]
 120 PLOT X,Y
 130 DRAW L,0: DRAW 0,-L: DRAW -L
,0: DRAW 0,L
 140 RETURN
 150
 160 REM PROC KWADRATY
 170 REM [S,X,Y,L]
 180 GO SUB PROC KWADRAT: REM [X,
Y,L]
 190 IF L<10 THEN GO TO 220
 200 LET X=X+S: LET Y=Y-S: LET L=
L-2*S
 210 GO SUB PROC KWADRATY: REM [S
,X,Y,L]
 220 RETURN
```

(IY + 71) = 0 lub po średniku, kiedy bit 7 = 1. Z = 0 świadczy, że nie znaleziono listy.

ZMNNUM (2550—2660) — wykorzystana w procedurze PRZEKAZ do uaktualniania numerów linii i instrukcji przez zamianę wartości zmiennych systemowych PPC i NEWPPC oraz SUBPPC i NSPPC.

SKLD (2690—2900) — wykorzystana w procedurze PRZEKAZ do kontroli składni w deklaracjach kolejnych zmiennych na liście parametrów. Dopuszczane są za nazwą zmiennej separatory ; ,] oraz typy (), $, $ () + separator. Na wyjściu w HL — adres za separatorem, a w A — kod separatora, w C bity 7—5 typ zmiennej, bity 4—0 nr pierwszej litery nazwy zmiennej. W przypadku błędu komunikat *Nonsense in BASIC*.

TWORZ (2930—3130) — wykorzystana w procedurze PRZEKAZ do tworzenia lokalnego obszaru zmiennych i pamięci pomocniczej lokalnej.

LIKW (3160—3300) — wykorzystana w programie głównym do likwidacji obszaru zmiennych lokalnych. Bit 0, (IY + 71) = 0 wskazuje, że w momencie wywołania procedury obsługi błędu, program korzystał z lokalnego obszaru zmiennych, jeżeli nie, najpierw następuje powrót do poziomu lokalnego (LPOZ 3440—3470). CY = 0 na wyjściu oznacza osiągnięcie poziomu podstawowego.

PPOZ (3330—3410) — wykorzystywana przez procedurę PRZEKAZ umożliwia przejście do niższego obszaru zmiennych. Adres początku obszaru zmiennych lokalnych przechowuje czasowo w VARST.

POCZMŁ i **POCZMP** — zmienne wykorzystywane do przechowywania adresów nazw aktualnie przepisywanej zmiennej odpowiednio w liście parametrów procedury i liście parametrów w instrukcji wywołania procedury.

Mam nadzieję, że PROCLOK będzie pomocny użytkownikom ZX Spectrum. Jeżeli zauważycie w trakcie użytkowania programu jakieś nie opisane tutaj reakcje, napiszcie o tym do „InforMika".



JACEK RAUK

Turbo Pascal 4.0: TPScreen — Moduł dodatkowych procedur ekranowych

Jednym z wielu powodów, dla których Turbo Pascal cieszy się wciąż rosnącą popularnością, jest możliwość pisania w nim programów nie tylko spełniających dokładnie wymagania użytkownika, ale też w pełni wobec niego „przyjacielskich". Instrukcje zawarte w module **Crt** pozwalają na uzyskanie szeregu efektów ułatwiających obsługę programu, np. migotania zwracającego uwagę na komunikat o błędzie:

```
program blink_error;
uses Crt;
begin
(...)
TextAttr  :=  TextAttr + Blink;
Write ('ERROR !');
TextAttr  :=  TextAttr – Blink;
(...)
end.
```

Nie znaczy to jednak, że korzystając z modułów standardowych **TP 4.0 (SYSTEM, CRT, DOS, GRAPH, PRINTER, GRAPH3, TURBO3)** wykorzystać możemy bezpośrednio wszystkie możliwości naszego komputera. Kłopoty sprawić może próba wprowadzenia do programu np. „rozwijających się" okienek z menu, które pojawiałyby się na ekranie, a następnie znikały odtwarzając „przykrytą" nimi treść. Teoretycznie można po „zamknięciu" okna wypisać ponownie utracone linie, nie jest to jednak ani wygodne, ani zawsze możliwe, gdyż w niektórych programach nie sposób z góry przewidzieć zawartości ekranu. Problem ten jest szczególnie istotny w programach interakcyjnych (np. edytory tekstu), w których zastosowanie tego sposobu komunikacji jest najkorzystniejsze. Przydatna byłaby oczywiście możliwość „przeczytania" przez program aktualnej zawartości przesłanianego przez okienko fragmentu ekranu i zapamiętania go, odczytania typu karty graficznej i trybu w jakim pracuje, wyświetlenia fragmentu ekranu „w negatywie" itp.; takich procedur jednak nie znajdziemy już w żadnym ze standardowych modułów. Listę podobnych „pobożnych życzeń" wobec **TP** można by przedłużać w nieskończoność.

Twórcy systemu **Turbo Pascal 4.0** pozostawili jednak na szczęście furtkę umożliwiającą rozwiązanie we własnym zakresie tych i podobnych problemów. Jest to bezpośrednie odwoływanie się do przerwań systemowych. Wystarczy znajomość numeru odpowiedniego przerwania i wymagań co do zawartości rejestrów proce-

sora, jakie ono stawia. Informacje te znaleźć można w szeregu pozycji literaturowych (np. *Peter Norton: Poradnik programisty komputerów osobistych IBM PC — tłumaczenie polskie krążące w postaci plików tekstowych*, czy też *Paweł Niezgodzki, Jerzy Rudowski: Przerwania systemowe, Klub użytkowników mikrokomputerów profesjonalnych, Zestaw II; Warszawa 1988*).

Problemy związane z tworzeniem menu oraz z innymi nietypowymi operacjami ekranowymi pomaga rozwiązać przerwanie nr 16 (**10h**). Oferuje ono wiele funkcji usługowych, z których zaledwie niektóre posiadają swoje odpowiedniki w postaci procedur i funkcji **TP 4.0**. Wyboru funkcji dokonuje się poprzez umieszczenie ich numeru w rejestrze **AH**. Dodatkowe parametry przekazywane są za pośrednictwem pozostałych rejestrów ogólnego przeznaczenia. Numery funkcji usługowych przerwania **10h** podaje Tabela 1.

Nie ma oczywiście większego sensu pisanie np. własnej wersji procedury **GotoXY** (moduł **Crt**) w oparciu o funkcję **02h**, czy funkcji **WhereX** i **WhereY** (również **Crt**) w oparciu o funkcję **03h**. Problematyczna też jest przydatność funkcji **04h** (ze względu na małą popularność piór świetlnych w sprzęcie klasy PC), praktyczne wydaje się jednak skorzystanie z niektórych innych funkcji.

W przedstawionym na **listingu 1** module **TPScreen** przedstawiłem wykorzystanie (bezpośrednio lub pośrednio) przerwania **10h** do:

- odczytu trybu pracy karty graficznej (funkcja **GraphMode**);
- odczytu maksymalnej liczby znaków w linii (funkcja **MaxVerSize**);
- odczytu typu karty graficznej (funkcja **GraphCard**);
- ustawienia rozmiaru kursora (procedura **Cursor**); (podobną procedurę opublikował *A. Kadlof — KOMPUTER* 11/88);
- odczytu znaku i jego atrybutu z dowolnego miejsca ekranu (procedura **GetChar**);
- zapisania znaku i jego atrybutu w dowolnym miejscu ekranu (procedura **PutChar**);
- wyświetlenia „odwrócenia" kolorów tła i znaku wybranego fragmentu ekranu (procedura **DisplnvStr**).

Ponadto, już bez uciekania się do przerwań, dodałem procedurę **Frame**, rysującą ramkę o zadanym położeniu. Szczegółowe omówienie wszystkich tych procedur byłoby zbyt nudne (niezbędne informacje podałem zresztą w formie komentarza). Poprzestanę na omówieniu działania procedur **GetChar** i **DisplnvStr**.

Działanie procedury **GetChar** rozpoczyna się od „zapamiętania" aktualnej pozycji kursora w zmiennych lokalnych **x1** i **y1** i przesunięciu kursora od pozycji określonej przez pierwsze dwa argumenty wywołania (**x** i **y**). Następnie przypisuje ona polom zmiennej lokalnej **reg** typu **Registers** odpowiednio wartości:

- polu **AH** numer funkcji usługowej (**08h**),
- polu **BH** numeru strony, na której dokonujemy operacji (tu: **0**).

Teraz procedura może już wywołać przerwanie **10h**, używając zmiennej **reg** jako drugiego parametru wywołania:

```
Intr ($10,reg);
```

Po wywołaniu przerwania procedura odczytuje kod znaku (**ASCII**) z pola **AL** zmiennej **reg**, zaś kod atrybutu z pola **AH**. Procedura **GotoXY** ustawia kursor z powrotem w pierwotnym położeniu.

Procedura **DisplnvStr** jest właściwie podwójną pętlą, w której kolejno wszystkie miejsca ekranu mieszczące się we wskazanych parametrami wywołania granicach (**x,y** — współrzędne lewego górnego rogu, **long** — „długość" inwersji, **high** — jej „wysokość") poddawane są kolejno następującym działaniom:

- odczytanie kodu znaku i jego atrybutu (**GetChar**);
- zmiana atrybutu tak, by kolor tła stał się kolorem znaku i na odwrót (realizowana w zależności od typu karty graficznej rozpoznanego przez funkcję **GraphCard**);
- zapisanie znaku ze zmienionym atrybutem (**PutChar**).

Procedura ta wykorzystuje więc pośrednio funkcje **08h**, **0Fh** i **09h**.

Mam nadzieję, że analiza pozostałych podprogramów w oparciu o ich treść i o komentarze nie sprawi nikomu kłopotu.

Zebranie procedur w oddzielny moduł ułatwia ich wykorzystywanie we własnych programach i skraca znacznie czas kompilacji. Zgodnie z regułami **Turbo Pascala** przyłączenie modułu **TPScreen** musi być poprzedzone włączeniem modułów **DOS** i **CRT**. Sposób wykorzystania opisanych procedur do tworzenia powstających na ekranie i znikających bez śladu komunikatów i menu, ilustruje przedstawiony na **listingu 2** program **Demo**.

Zastosowanie modułu **TPScreen** nie ogranicza się tylko do tworzenia menu. **Listing 3** przedstawia przydatny przy tworzeniu dokumentacji programów i instrukcji obsługi program rezydujący **StFPrint**. Jego działanie polega na zmianie sposobu obsługi przerwania **05h** (wywoływanego jednoczesnym wciśnięciem klawiszy **Shift** i **PrintScreen**) tak, by kopie ekranu tekstowego zapisywane były w pliku o nazwie podanej jako pierwszy parametr wywołania programu, lub w pliku **SCREEN.TXT** na aktualnej ścieżce aktualnego napędu dyskietek (gdy program wywołany został bez parametrów). Program ten również wykorzystuje opisany moduł.

Przykłady te nie wyczerpują zastosowań modułu. Przydać się on może np. przy pisaniu własnego edytora tekstu (zmiana wyglądu kursora w trybie **insert**, podświetlenie oznaczonego bloku itp.) oraz w szeregu innych programów. Można wzbogacić go o inne procedury obsługi ekranu, np. o procedurę **PrtScr** (*R. Wacławek — INFORMIK* 4/1987), czy też o napisaną samodzielnie procedurę do tworzenia okienek tekstowych podobnych do okienek graficznych pakietu **Borland Turbo Graphic v 1.0** (ciekawe ćwiczenie!). Mam nadzieję, że podany moduł przyda się przy tworzeniu programów nie tylko efektywnych, ale i łatwych w obsłudze.

Życzę przyjemnej pracy!

```
{              LISTING 1                }

{                                       }
{            TPScreen  v 0.9            }
{        modul ekranowych procedur      }
{              pomocniczych             }
{                      J.Rauk   1989    }
{                                       }

Unit                           TPScreen;
{***************************************}

               Interface
{***************************************}
uses
   Dos,Crt;
type
  Stat  =  0..2;
  {typ wykorzystany w procedurach       }
  { GraphCard i  Cursor                 }
const
  { GraphCard }               {  Cursor  }
   CGA   =  0;                OFF   = 0;
   EGA   =  1;                ON    = 1;
   HGC   =  2;                THICK = 2;

function               GraphMode: integer;
{ zwraca numer trybu pracy karty graf. }
{ tryb typ kolory rozdzielczosc karta   }
{ ------------------------------------- }
{  0    T   cz-b  40*25 srednia  CGA    }
{  1    T    16   40*25 srednia  CGA    }
{  2    T   cz-b  80*25 wysoka   CGA    }
{  3    T    16   80*25 wysoka   CGA    }
{  4    G     4   320*200 sred.  CGA    }
{  5    G    4 odc.320*200 sred.  CGA   }
{            szarosci                   }
{  6    G   cz-b  640*200 wys.   CGA    }
{  7    T   cz-b  80*25 wysoka MDA/HGC  }
{ 10    G     4   640*200 wys.   EGA    }
{ 13    G    16   320*200 sred.  EGA    }
{ 14    G    16   640*200 wys.   EGA    }
{ 15    G     4   640*350 b.wys. EGA    }

function             MaxVerSize : integer;
{ zwraca "szerokosc" ekranu tekstowego }
{ w znakach                             }

function                GraphCard: Stat;
{ zwraca typ karty graficznej           }

procedure      Cursor ( status : Stat );
{ procedura ustawia wyglad kursora      }
{ status = OFF   - kursor niewidoczny   }
{ status = ON    - kursor cienki        }
{ status = THICK - kursor pogrubiony    }
{ na podst.:   A.Kadlof KOMPUTER 11/88  }
procedure
 GetChar (x,y:integer;var chr,att:byte);
{ procedura odzytuje znak wskazany      }
{ wspolrzednymi x i y i zapisuje jego   }
{ kod w zmiennej chr, oraz kod jego     }
{ atrybutu w zmiennej att               }

procedure    PutChar (x,y,chr,att:byte);
{ procedura zapisuje znak o kodzie      }
{ chr i atrybucie att w miejscu         }
{ wskazanym wspolrzednymi x i y         }

procedure
      DispInvStr ( x,y,long,high:byte );
{ procedura odwraca kolory znakow  od   }
{ miejsca okreslonego przez wspolrz.    }
{             x,y                       }
{ do miejsca o wspolrzednych            }
{         x+long,y+high                 }

procedure    Frame (x1,y1,x2,y2 : byte);
{ rysuje ramke o wspolrzednych :        }
{ lewy gorny rog   - x1,y1;             }
{ prawy dolny rog  - x2,y2              }

             Implementation
{***************************************}

function               GraphMode: integer;
{***************************************}
  var
    reg  :registers; { pseudorejestry  }
  begin
    reg.AH:=$0F;
    Intr ($10,reg);
    GraphMode := reg.AL
  end;                       { GraphMode }

function             MaxVerSize : integer;
{***************************************}
  begin
    Case GraphMode of
      0,1          : MaxVerSize := 40;
      2,3,7        : MaxVerSize := 80;
      else           MaxVerSize :=  0;
    end
  end;                      { MaxVerSize }

function                 GraphCard: Stat;
{***************************************}
  var
    card : 0..2;
  begin
    Case GraphMode of
       1..6   : card :=   CGA;
          7   : card :=   HGC;
    10,13..15 : card :=   EGA;
    end;
    GraphCard := card
  end;                       { GraphCard }
 procedure Cursor        ( status : Stat );
{***************************************}
  var
    reg :registers;  { pseudorejestry  }
    att,atd :integer; { zmienne pomocn.}
  begin
    IF GraphCard=CGA then
      begin
        att := $607;
        atd := $308
      end
    else
      begin
```

```
        att  := $C0D;
        atd  := $50B;
      begin
        att := $C0D;
        atd := $50B;
      end;
    Case status of
      OFF   : reg.CH := $20;
      ON    : reg.CX := att;
      THICK : reg.CX := atd;
      else              EXIT;
    end;
    reg.AX := $100;
    Intr ($10,reg)
  end;                            { Cursor }

procedure
 GetChar (x,y:integer;var chr,att:byte);
{**************************************}
  var
    x1,y1 : byte;
    { poczatkowe wspolrzedne kursora   }
    reg :registers;  { pseudorejestry  }
  begin
    x1:=WhereX;
    y1:=WhereY;
    GotoXY(x,y);
    With reg do
      begin
        AH:=$08;
        BH:=0;
      end;
    Intr ($10,reg);
    With reg do
      begin
         chr:=AL;
         att:=AH
      end;
    GotoXY (x1,y1);
  end;                           { GetChar }

procedure    PutChar (x,y,chr,att:byte);
{**************************************}
  var
    x1,y1 : byte;
    { poczatkowe wspolrzedne kursora   }
    reg :registers;  { pseudorejestry  }
  begin

     x1:=WhereX;
     y1:=WhereY;
     GotoXY(x,y);
     With reg do
       begin
         AH:=$09;
         AL:=chr;
         BH:=0;
         BL:=att;
         CX:=1
       end;
     Intr ($10,reg);
     GotoXY (x1,y1);
   end;                          { PutChar }

 procedure
{**************************************}
```

```
    DispInvStr ( x,y,long,high:byte );
  var
    h,i,j,k,        { zmienne pomocnicze }
    lt,             { kod znaku          }
    at     : byte;  { kod atrybutu znaku }
  begin
   For h:=0 to (high-1) do
    For i:=0 to (long-1) do
      begin
        GetChar (x+i,y+h,lt,at);
        If GraphCard=HGC then
          If at=112 then
            at:=7
          else
            at:=112
        else
           begin
             j:=(at and 112) shr 4;
             k:=(at and 7 )  shl 4;
             at:= at and 136;
             at:= at or j or k
           end;
        PutChar (x+i,y+h,lt,at)
      end
  end;                        { DispInvScr }

procedure    Frame (x1,y1,x2,y2 : byte);
{**************************************}
  var
    x,y,i : integer; { zmienne pom.     }
  begin
    If (x2<=x1) or (y2<y1) then
      EXIT;
    x:=WhereX;y:=WhereY;
    If y1=y2 then
      begin
        GotoXY (x1,y1);
        For i:=x1 to x2 do
          Write (#205)                { ═ }
      end
    else
      begin
         GotoXY (x1,y1);
         Write (#201);                { ╔ }
         For i:=(x1+1) to (x2-1) do
           Write (#205);              { ═ }
         Write (#187);                { ╗ }
         For i:=(y1+1) to (y2-1) do
           begin
             GotoXY (x1,i);
             Write (#186);            { ║ }
             GotoXY (x2,i);
             Write (#186)             { ║ }
           end;
         GotoXY (x1,y2);
         Write (#200);                { ╚ }
         For i:=(x1+1) to (x2-1) do
           Write (#205);              { ═ }
           Write (#188);              { ╝ }
         end;
      GotoXY (x,y)
  end;                              { Frame }
end.                               { modulu }

{**************************************}
```

```
{                LISTING 2                 }

program                               DEMO;
{ demonstruje wykorzystanie biblioteki }
{ TPScreen do tworzenia MENU           }
  uses
   Dos,Crt,TPScreen;{dolaczenie modulow}
  type
    ScreenTab
      = array [1..80,1..25] of char;
    STPtr = ^ScreenTab;
  var
    I,J : integer; { liczniki }
    A   : STPtr;
    { wskaz na tabl. z "kopia" ekranu  }
    Ch,            { kod znaku         }
    Att : byte;    { kod atrybutu      }
  begin                      { PROGRAM }
{ ZAPELNIENIE EKRANU                   }
    For I:=1 to 24 do
      Writeln
        ('To jest linia nr':30,i:3);
    Delay (3000);
    New ( A );
{ UTWORZENIE  "OKNA"                   }
    Cursor ( OFF );
    For J:=2 to 5 do
      For I:=10 to 40 do
        begin
          GetChar ( I,J,Ch,Att );
          A^[I,J]:=Char (Ch);
          PutChar ( I,J,32,Att )
        end;
    Frame (10,2,40,5);
    GotoXY ( 11,3 );
    Write ( '    A tak moze' );
    GotoXY ( 11,4 );
    Write ( '    wygladac MENU' );
    Delay (3000);
{ PODSWIETLENIE  DOLNEJ LINII "OKNA"   }
    DispInvStr ( 12,4,17,1 );
    Delay (3000);
{ WYGASZENIE DOLNEJ LINII "OKNA"       }
    DispInvStr ( 12,4,17,1 );
{ PODSWIETLENIE  GORNEJ LINII "OKNA"   }
    DispInvStr ( 12,3,17,1 );
    Delay (3000);
{ ODTWORZENIE EKRANU                   }
    For J:=2 to 5 do
      For I:=10 to 40 do
        begin
          Ch:= Ord (A^[I,J]);
          PutChar ( I,J,Ch,Att );
          A^[I,J]:=' '
        end;
    GotoXY ( 1,25 );
    Delay (3000);
    Cursor ( ON );
    Dispose ( A )
  END.                     { PROGRAM }
{*************************************}
```

Tabela 1. Funkcje usługowe przerwania 10hex.

Nr funkcji dec	hex	Działanie
0	$00	Wybór trybu pracy
1	$01	Ustawienie rozmiaru kursora
2	$02	Ustawienie pozycji kursora na ekranie
3	$03	Odczytanie pozycji kursora
4	$04	Odczytanie pozycji pióra świetlnego
[illegible]	$05	Ustawienie aktywnej strony, dodatkowe operacje
6	$06	Przesuwanie okna w górę
7	$07	Przesuwanie okna w dół
8	$08	Odczytanie znaku i atrybutu
[illegible]	$09	Zapis znaku i atrybutu
10	$0A	Zapis znaku
11	$0B	Ustawienie palety kolorów
12	$0C	Zapis punktu
13	$0D	Odczyt punktu
14	$0E	Pisanie w trybie terminala (TTY)
15	$0F	Odczyt aktualnego trybu pracy
18	$12	Zarezerwowane dla karty EGA
19	$13	Dodatkowe procedury dla IBM PC AT

```
{                LISTING 3                  }

{$M 1024,0,000                              }
{ ustawienie  rozmiaru stosu i stery        }

program                            StFPrint;
{ zmienia obsluge przerwania $05 tak,       }
{ by kopia ekranu tworzona byla             }
{ w pliku o nazwie podanej jako             }
{ pierwszy parametr wywolania               }

  uses Dos,Crt,TPScreen;

  var
    pl : text;
    { plik zawierajacy "kopie" ekranu       }
    i  : integer; { licznik                 }

{$F+} { wymuszenie dalekich wywolan         }
  procedure  SaveScreen;        interrupt;
  { nowy sposob obslugi przerwania 05       }
    var
      i,j : integer;     { liczniki         }
      chr,               { kod znaku        }
      att  : byte;       { kod atrybutu     }
    begin
      Cursor ( OFF );
      Append ( pl );
      For j:=1 to 25 do
        begin
          For i:=1 to MaxVerSize do
            begin
              GetChar (i,j,chr,att);
              Write ( pl,Char(chr) )
            end;
          Writeln ( pl )
        end;
      For i:=1 to MaxVerSize do
        Write ( pl,#205 );
      Writeln ( pl );
      Close ( pl );
      Cursor ( ON );
    end;                       { SaveScreen }
{$F-} { wymuszenie bliskich wywolan         }

begin                             { PROGRAM }
  If ParamCount>0 then
    Assign ( pl,ParamStr(1) )
  else
    Assign ( pl,'Screen.TXT' );
  Rewrite ( pl );
  For i:=1 to MaxVerSize do
    Write ( pl,#205 );
  Writeln ( pl );
  Close ( pl );
  SetIntVec ( $05,@SaveScreen );
  Keep(0)
END.                              { PROGRAM }
{*****************************************}
```



OPERACJE NA DANYCH I ZBIORACH W MSX-BASIC

EDWARD KRAWCZYŃSKI

ROMAN KULA

Dialekt MSX-BASIC daje nam o wiele większe możliwości przetwarzania danych niż standardowy BASIC. Na uwagę zasługuje możliwość definiowania wielowymiarowych i wieloparametrowych tablic łańcuchowych i liczbowych, a także wieloparametrowych funkcji matematycznych i logicznych. Poniżej przedstawiamy te zagadnienia ilustrując je wieloma przykładami.

Wykorzystanie tablic w MSX-BASIC

Zamiast zmiennych prostych numerycznych lub łańcuchowych, których liczba jest ograniczona rozmiarami alfabetu i wskaźników użytych do ich oznaczenia, można używać zmiennych organizowanych w postaci tablic. Pamiętać należy, że MSX-BASIC rozróżnia tylko dwa pierwsze znaki oznaczające zmienną.

Zmienna tablicowa stanowi grupę lub macierz. Wszystkie zmienne posiadają tą samą nazwę. Do ich rozróżnienia służą indeksy określające położenie zmiennej w tablicy. Indeksy te określone są liczbami całkowitymi z deklarowanego w tablicy zakresu lub zmienną numeryczną.

Listing 1

```
10 DIM A(15)
20 K=17
30 DIM B(3,K)
40 DIM C$(3,K)
```

Powyższy przykład pokazuje sposób deklarowania różnego rodzaju tablic. Wiersz 10 rezerwuje pewien obszar pamięci dla tablicy liczbowej o nazwie „a", zawierającej 16 elementów o nazwach: a(0), a(1), .., a(14), a(15), zaś wiersz 30 dla tablicy liczbowej dwuwymiarowej 4 × 18-elementowej o nazwie „b" i następujących elementach:

```
b(0,0)  b(0,1)  b(0,2)  . . . .  b(0,16)  b(0,17)
b(1,0)  b(1,1)  b(1,2)  . . . .  b(1,16)  b(1,17)
  .       .       .     . . . .     .        .
  .       .       .     . . . .     .        .
b(3,0)  b(3,1)  b(3,2)  . . . .  b(3,16)  b(3,17)
```

Wiersz 40 deklaruje tablicę o identycznej strukturze oraz ilości elementów, lecz zawierającej zmienne łańcuchowe. Po zadeklarowaniu tablicy komputer nadaje automatycznie wartość 0 zmiennym tablicowym (numerycznym) lub łańcuch pusty o długości 0 spacji zmiennym łańcuchowym.

MSX-BASIC ogranicza każdy z wymiarów macierzy do 255, zaś liczba elementów macierzy ograniczona jest rozmiarami pamięci mikrokomputera. Oznacza to na przykład, że komputer odpowie raportem o przepełnieniu po ewentualnej deklaracji DIM a(5,300).

Tablica numeryczna bez deklaracji typu zmiennych jest automatycznie ustawiana na „podwójną precyzję". Jeśli zmienne w deklarowanych tablicach nie muszą mieć podwójnej precyzji to należy dodatkowo zadeklarować ich typ, np. DIM a%(3,9). Pozwala to zaoszczędzić miejsce w pamięci, a także przyspieszyć działanie programu. Poniższe zestawienie obrazuje zależność obszaru pamięci zajmowanego w zależności od typu zmiennych i tablic.

typ zmiennej	**ilość bajtów**
całkowita	2
pojedynczej precyzji	4
podwójnej precyzji	8
typ tablicy	**ilość bajtów**
całkowita	2 na element
pojedynczej precyzji	4 na element
podwójnej precyzji	8 na element

Deklaracja tablic lub zmiennych łańcuchowych powoduje zajęcie 3 bajtów przez nazwę tablicy lub zmiennej oraz obszar zależny od długości łańcucha.

W MSX-BASIC możliwe jest odwołanie się do nie zadeklarowanej tablicy. Można to uczynić np. zleceniem

D(1,3,2) = 153

które automatycznie powoduje zadeklarowanie tablicy liczbowej trójwymiarowej 11 × 11 × 11 elementowej o nazwie „D".

Równoważne to jest zleceniu DIM D(10,10,10), przyporządkowaniu wszystkim zmiennym z tablicy „D" wartości 0 oraz zmiennej D(1,3,2) wartości 153. Jeśli jednak zamierzamy korzystać z tablicy o rozmiarach mniejszych należy deklarować je za pomocą DIM, co pozwoli oszczędzić pamięć. Powyższe rozważania zilustruje następujący przykład:

Listing 2

```
10 D(1,2,3)=153
20 PRINT D(1,2,3)
30 PRINT D(2,2,3)
40 PRINT D(10,10,10)
50 PRINT D(11,10,10)
```

MSX-BASIC umożliwia także deklarowanie jedną instrukcją DIM kilka tablic, np.:

DIM A(3,4), B(5), C(2,12,7)

Próba deklaracji tablicy o nazwie już zadeklarowanej (bez względu na wymiar) jest niedopuszczalna — komputer zakomunikuje o błędzie. Aby taka deklaracja była możliwa należy wcześniej wymazać tablicę o danej nazwie z pamięci mikrokomputera.
Funkcja ta realizowana jest przez zlecenie

ERASE a

likwidujące w pamięci komputera tablicę numeryczną o nazwie „a" bez względu na jej strukturę i wymiary. Podobnie kasujemy tablicę zmiennych łańcuchowych wstawiając do zlecenia nazwę tablicy ze znakiem „$". Jednym zleceniem można również usunąć większą liczbę tablic z pamięci mikrokomputera np.

ERASE a,b,c,d
ERASE a$,b$,c$
ERASE a$,b,c,d$

Przedstawianie i używanie liczb i zmiennych w różnych systemach liczbowych

Komputery MSX standardowo obrazują wyniki wykorzystując dziesiętny system liczbowy. Oczywiście pracują wykorzystując system dwójkowy. Język MSX-BASIC posiada zlecenia, które pozwalają przedstawiać liczby i zmienne w systemach:

— dziesiętnym,
— dwójkowym (binarnym),
— ósemkowym,
— szesnastkowym (heksadecymalnym).

System dziesiętny najczęściej wykorzystywany do przedstawiania efektów działań nie wymaga większych objaśnień. Na uwagę zasługuje możliwość przedstawiania liczb z podwójną precyzją, z pojedynczą precyzją oraz całkowitych zmiennoprzecinkowych.

Aby zapisać lub zobrazować liczbę w systemie dwójkowym należy poprzedzić ją kodem &B, np. zlecenie
x = &B10110:PRINT x
da w efekcie na ekranie liczbę 22. Następuje więc przeliczenie liczby wyrażonej w systemie dwójkowym na dziesiętny.
Odwrotne przeliczenie następuje po użyciu zlecenia BIN$, np. PRINT BIN$ (22)
spowoduje wyświetlenie na ekranie wartości 10110 (dwójkowo).

Aby zapisać lub zobrazować liczbę w systemie ósemkowym należy poprzedzić ją kodem &O, np.

PRINT &O 120

spowoduje wyświetlenie liczby 80. Przeliczenie odwrotne następuje po użyciu zlecenia OCT$. Tak więc

PRINT OCT$ (80)

spowoduje wyświetlenie na ekranie wartości 120 (ósemkowo).

Analogicznie zapis liczby w systemie szesnastkowym wymaga użycia kodu &H np.

PRINT &H 1AF

spowoduje wyświetlenie liczby 431. Odwrotne przeliczenie następuje po użyciu zlecenia HEX$. Więc zapis

PRINT HEX$ (431)

spowoduje wyświetlenie na ekranie wartości 1AF (szesnastkowo).

Oczywiste jest, że możliwe jest przeliczanie dowolnych liczb z dowolnego wymienionego systemu na inny.

Funkcje, operatory logiczne i matematyczne, definiowanie funkcji

Zapisując wyrażenia arytmetyczne wykorzystujemy następujące operatory działań:

+ dodawanie
− odejmowanie
^ potęgowanie
∗ mnożenie
/ dzielenie
\ dzielenie całkowitoliczbowe (część całkowita ilorazu).

W wyrażeniach arytmetycznych można używać nawiasy zarówno okrągłe, jak i kwadratowe — także jedne zawarte w drugich. Wyrażenia, stałe i zmienne można porównywać używając do tego celu operatorów relacji, np.

X1<>10.2   X1>=X2
AS="MSX"   X>(A^2−C)∗(B+D)

Operatorów można również użyć do oceny prawdziwości wyrażenia, przy czym prawda jest wyświetlana jako −1 (minus jeden), zaś fałsz jako 0, np.

PRINT 2=1+4   da wartość 0
PRINT 5>2^2   da wartość −1

W wyrażeniach logicznych można używać następujących operatorów logicznych: NOT, AND, OR, XOR, EQV, IMP testujących różne relacje lub operacje.

Wszystkie wyżej wymienione operatory mają przypisany tzw. priorytet, czyli pierwszeństwo działania. Najwyższy priorytet ma potęgowanie, a następnie kolejno: „−" (zmiana znaku liczby), mnożenie i dzielenie, dodawanie i odejmowanie, operatory relacji, operatory logiczne w kolejności wymienionej wyżej.

MSX-BASIC posiada liczną grupę funkcji standardowych rezydujących w ROM. W Tabeli 1 zestawiono ich pełny wykaz wraz z objaśnieniem.

Przedstawiony zestaw funkcji standardowych pozwala na tworzenie innych funkcji, w tym również bardzo złożonych, np.

ASN(X)=ATN(X/SQR(1−X^2))   arcus sinus
HSN(X)=(EXP(X)−EXP(−X))/2   sinus hiperboliczny

Często w programowaniu zachodzi konieczność wielokrotnego obliczania wartości funkcji o złożonej postaci dla różnych wartości argumentów. W takich przypadkach najprostszym i najekonomiczniejszym rozwiązaniem jest skorzystanie z możliwości definiowania postaci funkcji wg schematu:

DEF FN nazwa (lista argumentów) = definicja funkcji

gdzie:

nazwa to zestaw znaków rozpoczynający się od litery (znaczące są tylko dwa pierwsze znaki np. A, A1, AB),
lista argumentów zaś obejmuje nazwy zmiennych (oznaczenia argumentów) rozdzielonych przecinkami — dopuszczalna liczba argumentów wynosi dziewięć;

definicja funkcji stanowi wyrażenie prezentujące postać funkcji. Należy pamiętać że DEF FN ... nie może być użyte w trybie bezpośrednim. Po zdefiniowaniu funkcji każdorazowo, gdy chcemy znać lub wykorzystywać jej

wartość dla określonych argumentów, należy odwołać się do niej wg schematu:

FN nazwa (lista wartości argumentów)

gdzie:

lista wartości argumentów przedstawia ciąg liczb lub zmiennych w ilości zgodnej z ilością argumentów w DEF FN, np.

Listing 3

```
10 DEF FN F(X,Y)=X+2*Y

20 A=1

30 K=FN F(A,2)

40 PRINT "K=";K
```

Realizacja tego przykładu da w wyniku k = 5.

Przedefiniowanie postaci funkcji nie wymaga uprzedniego wymazania poprzedniej postaci funkcji. Zlecenia CLEAR i RUN czyszczą z pamięci mikrokomputera zmienne zdefiniowane przez funkcję.

Jeśli chcemy, aby argumenty funkcji miały określony typ przed obliczeniem wartości funkcji, należy w tym celu dokonać ich określenia wykorzystując zlecenia DEF INT, DEF SNG lub DEF DBL.

Operacje na zbiorach danych, wprowadzanie danych

Większość wykonywanych programowo działań polega na obliczaniu wartości wyrażeń lub podstawianiu pod zmienne parametrów lub wartości liczbowych. Do tego celu służy instrukcja podstawiania LET o postaci:

Listing 4

```
10 LET A=7

20 B=5

30 LET C=2*A+B

40 A$="Ala ":B$="ma ":C$="kotów"

50 D$=A$+B$

60 RRINT D$;C;C$

70 LET E="msx"
```

Wykonanie linii 70 spowoduje wyświetlenie komunikatu o błędzie „Type mismatch", jako że typ zmiennej i wyrażenia w instrukcji musi być zgodny.

Jednym ze sposobów nadawania wartości zmiennym jest wprowadzanie ich przez obsługującego komputer z klawiatury w trakcie biegu programu. Umożliwia to instrukcja programowego wprowadzania danych o postaci:

INPUT "komentarz"; lista zmiennych,

gdzie komentarz stanowi dowolny zestaw znaków.

Po napotkaniu tej instrukcji komputer wstrzymuje realizację programu aż do wprowadzenia przez obsługę wszystkich zmiennych z uwzględnieniem ich typu i ilości.

Oczekiwanie na wprowadzenie danych sygnalizowane jest wyświetlaniem znaku „?" lub komentarza i znaku

TABELA 1

FUNKCJA	WYNIK
SIN (X) COS (X) TAN (X)	Wartość funkcji sinus, cosinus, tangens z argumentu podanego w mierze łukowej, a wynik obliczeń podawany jest z podwójną precyzją.
ATN (X)	Wartość funkcji arcus tangens, wynik zawiera się w przedziale $[-\pi/2,\pi/2]$ i jest podawany z podwójną precyzją.
SQR (X)	Wartość pierwiastka kwadratowego z dodatniego argumentu.
LOG (X)	Wartość logarytmu naturalnego z dodatniego argumentu.
EXP (X)	Wartość wyrażenia: e^X, X< = 146.
SGN (X)	Znak liczby, wartość 1, 0 lub −1 w zależności od znaku argumentu.
ABS (X)	Wartość bezwzględna argumentu.
INT (X)	Największa liczba całkowita nie większa od argumentu.
FIX (X)	Część całkowita argumentu.
RND (X)	Liczba pseudolosowa z przedziału <0;1).
LEN (A$)	Liczba znaków (długość) łańcucha A$.
LEFT$ (A$,X) RIGHT$ (A$,X)	Łańcuch X-znakowy zawarty w łańcuchu A$ licząc od lewego (LEFT) lub prawego (RIGHT) końca łańcucha A$.
MID$ (A$,X,Y)	Łańcuch Y-znakowy pochodzący z łańcucha A$ rozpoczynający się elementem znajdującym się na pozycji X licząc od lewej strony tego łańcucha.
INSTR (X,A$,B$)	Pozycja licząc od lewej w łańcuchu A$, gdzie łańcuch B$ zawiera się w A$, zaś X wskazuje pozycję znakową, od której rozpocznie się przeszukiwanie łańcucha A$. Instrukcja rozróżnia małe i duże litery.
CHR$ (X)	Łańcuch, którego kodem ASCII jest argument.
SPACE$ (X)	Łańcuch składający się z X spacji.
SPC (X)	Wyświetla X znaków pustych (tylko pod PRINT i LPRINT).
STR$ (X)	Tworzy ze zmiennych numerycznych X zmienne łańcuchowe, na których można wykonywać operacje łańcuchowe.
STRING$ (X,Y) STRING$ (X,A$)	Łańcuch X-znakowy o kodzie ASCII równym Y lub o kodzie pierwszego znaku łańcucha A$.
ASC (A$)	Wartość kodu ASCII dla pierwszego znaku łańcucha A$.
BIN$ (X) OCT$ (X) HEX$ (X)	Podaje odpowiednio zapis dwójkowy, ósemkowy, szesnastkowy argumentu dziesiętnego.
FRE (0)	Ilość wolnej pamięci.
FRE ("")	Ilość wolnej pamięci dla łańcuchów.
DSKF (0)	Ilość wolnej pamięci na dysku.
LPOS (X)	Bieżąca pozycja w wierszu głowicy drukującej drukarki.
POS (X)	Bieżąca współrzędna pozioma kursora.
CSRLIN	Bieżąca współrzędna pionowa kursora.
CINT (X) CSNG (X) CDBL (X)	Konwersja argumentu na liczbę całkowitą, pojedynczej precyzji, podwójnej precyzji.

„?”. Kolejny przykład pokazuje warianty wykorzystania instrukcji INPUT.

Listing 5

```
10 CLS
20 PRINT "a=";:INPUT A
30 INPUT "b=";B
40 PRINT"wprowadź A$,k";:INPUT A$,K
50 INPUT"Podaj d,e,f ";D,E,F
60 FOR I=1 TO 10
70 PRINT"a(";I;")=";:INPUT A(I)
80 NEXT I
90 PRINT "a(3)=";A(3)
```

Dla zmiennych łańcuchowych możemy używać instrukcji o postaci ogólnej:
LINE INPUT [”komentarz”;] lista zmiennych łańcuchowych, która przyjmuje z klawiatury łańcuch o długości do 254 znaków oraz nie generuje pytajnika jak INPUT.
Z kolei instrukcja

INPUT$ (X)

przyjmuje określoną argumentem X ilość znaków z klawiatury zapamiętując je, a następnie przechodzi do dalszej realizacji programu.

Listing 6

```
10 CLS
20 PRINT "Naciskaj różne klawisze"
30 PRINT INPUT$(3)
40 B$=INPUT$(2)
50 PRINT B$
60 C$=INPUT$(1)
70 IF C$="1" THEN PRINT"Nacisnąłeś 1"
80 IF C$<>"1" THEN PRINT"Nacisnąłęś nie 1"
90 GOTO 60
```

Instrukcja:

INKEY$

tworzy jednoznakowy łańcuch odpowiadający naciśniętemu klawiszowi. Łańcuch ten może być pusty, jeśli nie naciśnięto żadnego klawisza lub kilka naraz.

Listing 7

```
10 CLS
20 PRINT"Program będzie kontynuowany gdy
naciśniesz dowolny klawisz"
30 IF INKEY$="" THEN 30 ELSE 40
40 K$=INKEY$
50 PRINT K$
60 GOTO 40
70 REM Sprawdź działanie programu umieszczając
w linii 50 po K$ różne separatory (przecinek,
średnik)
```

Instrukcja:

SWAP zmienna,zmienna

wymienia wzajemnie między sobą obie zmienne tego samego typu. Próba wymiany między sobą zmiennej numerycznej na łańcuchową powoduje błąd.
Instrukcje:

DATA lista stałych

oraz

READ lista zmiennych

umożliwiają wprowadzanie dużej liczby zmiennych numerycznych lub łańcuchowych. Występują one łącznie, jak np. FOR-NEXT. DATA przechowuje dane, które w trakcie wykonywania programu będą podstawiane pod zmienne przy pomocy READ. Widać stąd, że lista stałych oraz lista zmiennych muszą być komplementarne. Koniecznym warunkiem jest więc, aby liczba zmiennych w READ nie przekraczała liczby stałych w DATA oraz by typ zmiennej odpowiadał typowi stałej. MSX dopuszcza jednak możliwość niespełnienia tego ostatniego warunku. Pokazuje to poniższy przykład.

Listing 8

```
10  DIM A(8),A$(12)
20  D$="dana"
30  DATA 0,1,2,ala,4,"bela",5,d$,6,7,8
40  FOR I=0 TO 6
50  READ A(I)
60  READ A(I$)
70  PRINT " A(";I;")=";A(I)
80  PRINT "A$(";I;")=";A$(I)
90  NEXT I
100 PRINT "Naciśnij dowolny klawisz!"
110 IF INKEY$="" THEN 110 ELSE 120
120 GOTO 20
130 DATA 9,10,11,12,13,14,15,16,17,18,19,
20,21,22,23,24,25,26,27,28,29,30
140 REM dopisz linię 35 RESTORE a
następnie 35 RESTORE 130
```

Przykład ten pokazuje również sposób użycia instrukcji

RESTORE

oraz

RESTORE nr linii.

Powoduje ona powtórne odczytanie listy danych od początku tej listy (RESTORE) lub od danych zawartych w linii wskazanej przez RESTORE nr linii.



Router, czyli o kierowaniu ruchem

Przy obecnym poziomie informatyzacji polskich przedsiębiorstw, niemal każdy komputer klasy IBM PC jest wykorzystywany do kilku różnych zastosowań, często też korzysta z niego kilka osób. Regułą jest, że na dysku twardym znajduje się wiele programów aplikacyjnych. Zwykle też użytkownicy tworzą odrębne podkatalogi dla swoich programów. Niestety, przejście od pracy z jednym programem do pracy z innym, umieszczonym w innym podkatalogu, wymaga nużącej sekwencji poleceń **cd** (ang. *Change Directory*) i podawania nowych ścieżek dostępu. Utrudnione jest także wywoływanie, z dowolnego podkatalogu, programów pomocniczych, takich jak np. XTREE.

Trudnościom tym można zaradzić, konstruując specyficzną strukturę podkatalogów (najlepiej przy pomocy programu XTREE) oraz wykorzystując procedury wsadowe (**batch**). Struktura podkatalogów powinna odwzorowywać podział tematyczny zawartości dysku: podkatalogi pierwszego stopnia — grupy programów pokrewnych (np. „Edytory"), drugiego stopnia — programy (np. „Chiwriter", „Wordstar" itp.), a podkatalogi trzeciego stopnia — biblioteki, zbiory robocze itp. W takiej strukturze każdy program powinien posiadać swój własny katalog. Jest to użyteczne, ponieważ pracując np. z XTREE można łatwo orientować się w zawartości dysku (nazwy większości plików wykonywalnych różnią się od tytułu programu, natomiast katalog można nazwać tytułem). Porządkując pliki (np. kasując zbędne kopie robocze) jesteśmy pewni, że omyłkowo nie zniszczymy istotnych fragmentów innego programu. Ostatnią, lecz wcale niebagatelną korzyścią z takiej, pozornie dość skomplikowanej, organizacji dysku jest większy repertuar dostępnych nazw zbiorów: w różnych katalogach mogą występować te same nazwy.

Opisana struktura katalogów nie wystarcza sama do usprawnienia pracy, wręcz przeciwnie, komplikuje ją. Dla pełnego komfortu trzeba jeszcze skorzystać z dwóch usług systemu MS-DOS: polecenia **PATH** i plików wsadowych **.BAT**.

Polecenie **PATH** służy do poinformowania systemu operacyjnego, gdzie szukać plików wykonywalnych (typu .EXE, .COM lub .BAT), jeśli nie ma ich w bieżącym podkatalogu. Polecenie ma format:

PATH = *ścieżka*_1;*ścieżka*_2;...

Jeśli pliku o podanej nazwie nie ma w bieżącym katalogu, to system szuka pliku w katalogach wyspecyfikowanych podanymi ścieżkami dostępu (w kolejności ich podania). Wystarczy więc wskazać ścieżki dostępu do wszystkich podkatalogów (najlepiej w kolejności odpowiadającej popularności programów), aby system bez problemu odnajdywał wywoływane pliki.

Opisane rozwiązanie spełnia swoje zadania tylko w przypadku programów mieszczących się w jednym pliku. Niestety, większość programów zawiera biblioteki — np. krojów czcionki, procedur sterujących (ang. *driver*) — i oczekuje, że wszystkie te zbiory znajdują się w bieżącym podkatalogu. Wywołanie (przy pomocy polecenia **PATH**) z innego katalogu prowadzi więc do nieuchronnego komunikatu o błędzie.

Na szczęście istnieją procedury wsadowe **batch**. Dla każdego programu użytkowego należy utworzyć procedurę wsadową, która zmieni katalog na właściwy i wywoła plik danego programu. Procedura wsadowa może też zawierać wywołania programów rezydujących i sterujących np. myszą. A więc plik batch inicjujący pracę systemu OrCAD może wyglądać następująco:

echo off
cd ORCAD
mmouse
draft

Do utworzenia plików wsadowych można posłużyć się edytorem systemowym (przykładowe polecenie: **copy con orcad.bat**) lub dowolnym innym, np. funkcją „Notes" SideKicka. Wszystkie procedury wsadowe wygodnie jest zgromadzić w oddzielnym pliku podkatalogu (np. pod nazwą ROUTER) i wpisać do pliku AUTOEXEC.BAT polecenie

PATH = C:\;\ROUTER

Pierwsza ścieżka wskazuje na katalog główny, w którym powinny znaleźć się podstawowe programy narzędziowe (np. PARK), a druga — na katalog procedur wsadowych „kierujących ruchem". Od tej chwili każdy program może być wywoływany z dowolnego podkatalogu przez proste wpisanie jednej nazwy.

Winchester pod lupą

Upowszechnienie i potanienie twardych dysków sprawiło, że użytkownicy zaczęli traktować je jako szybki i wygodny magazyn danych. Gdy jednak zgromadzone informacje mają dużą wartość, zaufanie do Winchesterów maleje. Nawet produkty renomowanych firm, eksploatowane według wszelkich reguł sztuki, sprawiają czasem użytkownikowi wstrząsającą niespodziankę w postaci komunikatu DATA ERROR, READ FAULT, SECTOR NOT FOUND lub nawet, HARD DISK FAILURE. Przyczyn takich zdarzeń może być kilka. Pierwszą z nich jest „rozjeżdżanie się" ścieżek na dysku. Nowy dysk sformatowany fabrycznie ma wąskie, wyraźne ścieżki z wpisanymi identyfikatorami ścieżek i sektorów. Temperatura, wibracje, zmęczenie materiału i temu podobne czynniki sprawiają, że kontur ścieżki zaczyna się rozmywać: dane zapisane rano, gdy mechanizm pozycjonowania głowicy był zimny, są umieszczone bliżej środka, a informacje południowe bliżej krawędzi dysku. Im dłużej eksploatujemy dany fragment dysku, tym szerszy jest zapis, aż wreszcie ścieżki zaczynają na siebie nachodzić i danych nie można już odczytać. Co więcej, każda kolejna próba zapisu na takiej ścieżce kończy się fiaskiem.

Kolejna przyczyna to nieuchronne starzenie się warstwy magnetycznej dysku. Nie chodzi bynajmniej tylko o zużycie mechaniczne. Najgroźniejsze są zmiany właściwości fizycznych: gładkości i zdolności magnesowania. Te cechy zmieniają się z czasem, podlegają wahaniom związanym ze zmianami temperatury i zależą od jakości pokrycia podłoża warstwą magnetyczną. Po pewnym czasie nawet na najstaranniej wyprodukowanym dysku dane mogą „zatrzeć się".

Najgroźniejszym wrogiem danych na dysku są zakłócenia elektryczne pochodzące z sieci lub powodowane przez ładunki elektrostatyczne. Rozmycie ścieżek lub rozmagnesowanie nośnika zdarza się o wiele rzadziej, niż przepięcia i spadki napięć zasilających. Mimo starannie projektowanych zabezpieczeń sterownikom dyskowym zdarza się zapisać w takich chwilach przypadkowe dane w przypadkowym miejscu dysku, niestety czasem także na tablicy alokacji lub katalogu.

Zabezpieczenie się przed podobnymi przygodami polega zwykle na kopiowaniu zawartości twardego dysku na dyskietki lub kasetę streamera. Jest to jednak zajęcie czasochłonne, uciążliwe, a w dodatku nie daje pełnych rezultatów — mało kto tworzy awaryjne kopie codziennie, a i wtedy rezultat całodziennej pracy może być zniweczony zapisaniem danych na rozmytej ścieżce. O cenie streamera lepiej nie wspominać...

Odmienną, interesującą metodę zapobiegania kłopotom z twardymi dyskami zaproponowała firma Prime Solutions Inc. z San Diego oferując program **Disk Technician**. Jest on przedstawicielem nowej gałęzi oprogramowania użytkowego — programów konserwacyjno-naprawczych, wykorzystujących techniki sztucznej inteligencji. Disk Technician sprawdza jakość zapisów na twardym dysku i przeprowadza kompleksową naprawę wykrytych usterek. Jeśli stwierdzi, że któraś ze ścieżek rozmywa się, odczytuje z niej dane, składa je w pamięci komputera, a następnie ponownie formatuje całą ścieżkę przywracając jej wyraźny kontur. Formatowanie odbywa się według specjalnego algorytmu, zapisywana jest od nowa cała ścieżka łącznie z identyfikatorami sektorów. Nowo zapisana ścieżka jest szczegółowo testowana — tylko po całkowitym sprawdzeniu dane wracają na swoje dotychczasowe miejsca. Jeśli powierzchnia dysku okaże się uszkodzona, dane są przenoszone w inne miejsce dysku, a wadliwy obszar jest blokowany dla dalszego wykorzystania przez DOS.

Nie tylko ścieżki rozmyte mogą być naprawione przez Disk Technician. Każdy pojedynczy bit — zapisany czy nie — jest sprawdzany pod kątem jakości zapisu. Wykryte błędy — niepewny poziom namagnesowania, zbyt duża „twardość" materiału magnetycznego itp. wyzwalają kompleksowy proces naprawy wadliwego obszaru, zakończony odtworzeniem jego zawartości lub przeniesieniem jej na inne miejsce.

Jeszcze jedną, istotną cechą programu jest moduł **SafePark**. Moduł ten instaluje się samoczynnie na twardym dysku, a następnie każdorazowo ładuje się do pamięci wraz z systemem operacyjnym. Jego zadaniem jest przemieszczanie głowicy nad bezpieczny obszar „lądowiska" przy każdej przerwie między operacjami dyskowymi trwającej dłużej niż ok. 7 sekund (czas ten może być zmieniany przez użytkownika). Wstrząsy i przypadkowe zapisy przestają wówczas zagrażać istotnym danym.

Program Disk Technician sprzedawany jest w cenie 99,95 dol., wymaga komputera z systemem operacyjnym PCNS-DOS 2.1 lub nowszym, 256 K pamięci RAM i jednym napędem dyskowym 5,25 cala (oczywiście także z dyskiem sztywnym). Obsługa nie wymaga żadnych umiejętności — wystarczy włożyć dyskietkę i włączyć komputer. Codzienne użycie programu wymaga około minuty — znacznie mniej niż skopiowanie dysku na kasetę streamera. Wygląda na to, że nareszcie komputery zaczynają dbać o siebie!

Komputerowy rozmówca

Nowoczesne biuro to nie tylko komputery, ale również sprawne środki komunikacji między ludźmi i między maszynami. Wprawdzie w polskich realiach modemy, aparaty telekopiowe (telefaksy) i automaty zgłoszeniowe (automatyczne sekretarki) są jeszcze rzadko spotykane, lecz każdy chyba użytkownik marzy o posiadaniu wszystkich tych urządzeń na swoim biurku. Gdy marzenia te się spełnią pojawia się kolejny problem: jak sprawiedliwie podzielić między wszystkie funkcje komunikacyjne jedną linię telefoniczną. Najlepiej byłoby połączyć wszystkie urządzenia w jedno wraz ze zwykłym telefonem, całość „upchnąć" we wnętrzu komputera i dołączyć do telefonicznego gniazdka w ścianie...

Okazuje się, że nie jest to bynajmniej futurystyczny projekt: takie właśnie zalety posiada zintegrowana karta komunikacyjna **Orator Link** oferowana przez firmę Atlantic Office Communications. Stanowi ona połączenie modemu, automatycznej sekretarki i zwykłego telefonu, a ponadto posiada wiele interesujących funkcji pomocniczych.

Orator Link ma postać typowej karty rozszerzającej PC — jedyne różnice to zamykająca cały pakiet, plastykowa obudowa i dwa telefoniczne gniazdka na tylnej ściance karty. Jedno z tych gniazdek należy połączyć z linią telefoniczną, a drugie z aparatem telefonicznym. Po zainstalowaniu karty we wnętrzu komputera można zwyczajnie korzystać z telefonu, bez względu na to, czy komputer jest włączony, czy nie. Orator przejmuje bowiem kontrolę nad linią telefoniczną tylko wtedy, gdy tego potrzebuje. Przygotowania do pracy kończy zainstalowanie programu sterującego **Orator Manager** (na czystej dyskietce lub na twardym dysku). Procedury instalacyjne tworzą właściwe pliki AUTOEXEC.BAT i CONFIG.SYS lub modyfikują już istniejące.

Po wywołaniu program **Orator Manager** pozostaje stale w pamięci operacyjnej i pracuje „w tle". W dowolnej chwili, podczas pracy z innymi programami, można przywołać Oratora na pierwszy plan przyciskając jednocześnie lewy klawisz **Shift**, **Ctrl** i **Alt**. Ta sama kombinacja klawiszy odsyła Oratora ponownie „w tło". Podczas pracy pierwszoplanowej Orator zapewnia bogaty interfejs z użytkownikiem, oparty na okienkach, rozwijanych menu i sterowany klawiszami funkcyjnymi.

Zasadniczym elementem karty **Orator Link** jest programowany procesor sygnałowy. W zależności od zastosowanego programu może on realizować funkcje modemu w standardach V21, V22, V22bis i V23, pracować z szybkościami transmisji 300, 1200 i 2400 bodów, a także w niesymetrycznym trybie 1200/75 bodów, wykorzystywanym przez brytyjskie publiczne służby informacyjne Prestel. Procesor sygnałowy służy także do obsługi komunikacji głosowej: pozwala zapisać sygnał mowy w postaci cyfrowej na dysku, jak również odtworzyć zapisaną na dysku wypowiedź. Mowa jest próbkowana z częstotliwością 8 kHz i kodowana według specjalnych metod kompresji. Pozwala to zapisać sekundę mowy na przestrzeni ok. 1 kilobajta. Uzupełnieniem sprzętowych możliwości Oratora jest zegar czasu rzeczywistego.

Użytkownik komputera wyposażonego w kartę **Orator Link** może utworzyć na dysku katalogi numerów telefonicznych. Ilość katalogów jest ograniczona tylko pojemnością dysku, a każdy z nich może zawierać 999 pozycji. Pojedyncza pozycja zawiera trzyliterowy kod identyfikacyjny, nazwę lub nazwisko abonenta, numer telefonu, a także informacje pomocnicze: czy wybieranie numeru jest realizowane dekadowo (impulsami) czy częstotliwościowo, czy Orator jest dołączony do linii miejskiej czy wewnętrznej, jaką cyfrę trzeba wybrać, by otrzymać dostęp do linii zewnętrznej, czy pod danym numerem należy się spodziewać człowieka czy komputera itd. W przypadku numerów służb informacyjnych i innych komputerów wpisane są też dane o wymaganym protokole, szybkości transmisji, sekwencji identyfikacyjnej (*log-on*). Numer z katalogu wybiera się przez podanie trzyliterowego identyfikatora lub podświetlenie odpowiedniej pozycji w menu.

Po zgłoszeniu się wywoływanego numeru Orator sam rozpoznaje, czy ma do czynienia z człowiekiem czy z innym komputerem. Jeśli zgłosi się komputer, to z katalogu pobierane są dane o parametrach transmisji, następuje inicjalizacja funkcji modemowych, wysłanie zdefiniowanej przez użytkownika sekwencji identyfikacyjnej i rozpoczęcie automatycznej wymiany danych. Jeśli zgłosi się człowiek Orator wezwie odpowiednim

CIEKAWE KSIĄŻKI

Ryszard K. KOTT, PROGRAMOWANIE W JĘZYKU PASCAL, Wydawnictwa Naukowo-Techniczne, Warszawa 1988. Nakład 30 000 egz., wydanie pierwsze, cena: zł 780.

Język **Pascal**, pomyślany przez swego twórcę, Niklausa Wirtha, jako li tylko narzędzie do nauki dobrego, strukturalnego programowania, zdobył sobie w ciągu ostatnich lat ogromną popularność. Pascal jest obecnie podstawowym językiem kształcenia informatyków, jest także językiem powszechnie używanym w publikacjach do opisu algorytmów, struktur danych, reguł postępowania, itp. Kompilatory Pascala dostępne są we wszystkich praktycznie instalacjach komputerowych, minikomputerowych i mikrokomputerowych. Większość poważnych firm produkujących oprogramowanie mikrokomputerów opracowała własne implementacje języka Pascal. Coraz łatwiejszy dostęp do komputerów osobistych sprawia, że znacznie poszerzył się krąg ludzi wykorzystujących te implementacje i uważających je za wygodne, nowoczesne i silne narzędzia programistyczne.

Na polskim rynku księgarskim ukazało się sporo pozycji omawiających poszczególne implementacje języka Pascal, dostępne są także książki zawierające opis języka, brakowało jednak podręcznika przeznaczonego do praktycznej nauki programowania w Pascalu.

Ryszard K. Kott podjął próbę stworzenia takiego podręcznika, czego efektem jest recenzowana książka.

Rozpoczyna ją omówienie wstępnych, podstawowych pojęć programowania (takich, jak algorytm, język programowania, program, programowanie). Nieco uwagi poświęcono procesom translacji i interpretacji programów, wyjaśniono różnice w rodzajach usług udostępnianych na poszczególnych poziomach programowania (sprzęt, system operacyjny, język wysokiego poziomu). Autor przekazał tu minimum wiedzy niezbędnej nowicjuszom nieobeznanym z techniczną stroną programowania. Przedstawienie dwóch metajęzyków definiowania języków programowania (upraszczając: formalnych sposobów zapisu definicji), t.j.: notacji Backusa-Naura (BNF) oraz diagramów składniowych kończy pierwszy, przeznaczony dla absolutnych nowicjuszy, rozdział podręcznika.

Bogato komentowane diagramy składniowe są konsekwentnie stosowane w pozostałych rozdziałach książki do definiowania wprowadzanych kolejno, w logicznym porządku, pojęć i konstrukcji języka Pascal. Graficzne zobrazowanie elementów języka i reguł składni ułatwia ich zrozumienie w takim stopniu, iż mogim zdaniem, nawet początkujący nie powinni mieć kłopotów z przyswojeniem prezentowanego materiału.

Naukę ułatwia też układ treści podręcznika — w pierwszej jego części wprowadzono tylko najprostsze elementy języka, omawiając je jednak od podstaw, gruntownie, a przy tym jasno i precyzyjnie. Lektura choćby tylko tej części powinna już umożliwić czytelnikowi samodzielne pisanie nieskomplikowanych programów w Pascalu.

Druga część podręcznika wprowadza pozostałe, bardziej złożone pojęcia oraz zawiera uzupełnienia i rozszerzenia, omówionych w części pierwszej fragmentów konstrukcji języka Pascal (całość opisu języka ściśle odpowiada międzynarodowej normie ISO nr 7185).

Książka Ryszarda K. Kotta w pełni zasługuje na miano podręcznika programowania. Praca zawiera bowiem nie tylko kompletny opis języka, ale także omówienia zastosowań (i ograniczeń stosowalności) kolejno wprowadzanych konstrukcji językowych oraz wskazania celów, dla realizacji których te konstrukcje są szczególnie użyteczne. Autor po-

sygnałem dźwiękowym operatora do podniesienia słuchawki i rozpoczęcia rozmowy — oczywiście za pośrednictwem procesora sygnałowego karty.

Kartę **Orator Link** można zaprogramować do obsługi połączeń przychodzących. Każde wywołanie pochodzące z innego komputera spowoduje nawiązanie automatycznej wymiany danych. Na telefon od człowieka Orator może zareagować odtworzeniem zapisanego na dysku komunikatu głosowego, a następnie zarejestruje (poprzez próbkowanie i kodowanie) wypowiedź rozmówcy na dysku opatrując ją etykietką z czasem i datą.

Wszystkie funkcje Oratora mogą być aktywowane przez wbudowany zegar o określonych, wstępnie zaprogramowanych porach. Można więc np. zlecić komputerowi, by o danej godzinie zadzwonił do określonej osoby i przekazał jej wiadomość syntetycznym głosem. Można też wykorzystywać nocne zniżki taryf telefonicznych, programując Oratora tak, by w środku nocy połączył się z innym komputerem i przekazał mu dane.

Zapisywanie komunikatów głosowych w postaci cyfrowej jest proste — wystarczy wybrać menu „Rejestracja wiadomości", podać nazwę pliku, wcisnąć klawisz spacji i zacząć mówić do mikrotelefonu. Natychmiast po nagraniu można je odsłuchać w słuchawce. Syntetyczna mowa Oratora jest wprawdzie lekko bulgocząca — tak jak np. głos nurka spod wody — lecz w pełni zrozumiała.

Karta **Orator Link** ma możliwość zdalnego sterowania. Wyposażony w nią komputer może o dowolnej porze połączyć się z innym komputerem posiadającym kartę Orator, a następnie korzystać z jego zasobów sprzętowych i plików jak ze swoich własnych. Realną perspektywą jest więc korzystanie z mocy obliczeniowej firmowego komputera z własnego, domowego PC po godzinach pracy. Ochrona przed niepowołanym dostępem jest zagwarantowana systemem haseł.

Wszystkie te cechy karty **Orator Link** sprawiają, że jest ona jedną z ciekawszych propozycji na rynku brytyjskim. Mimo że możliwość zapisu i odczytu mowy może jeszcze dziś wydać się egzotyczna, to jednak cena 795 funtów jest interesująca nawet wtedy, gdy traktuje się Oratora wyłącznie jako wielozadaniowy modem z działającym w tle oprogramowaniem komunikacyjnym. Czekamy więc na pierwsze Oratory homologowane i pracujące w Polsce.

Opracował: Janusz Wrześniak

---

dał sposoby tworzenia efektywnych i eleganckich algorytmów, zamieścił także uwagi dotyczące organizacji struktur danych. Godna pochwały jest konsekwencja, z jaką starał się wyrobić u czytelnika nawyki dobrego programowania.

Cały wykład jest bogato ilustrowany przykładami typowych zastosowań elementów języka Pascal. Na uwagę zasługuje sposób prezentacji przykładów: klarowne sformułowanie problemu, jasne (choć niekiedy zbyt skrótowe) przedstawienie algorytmu rozwiązania i wreszcie takst w Pascalu — egzemplifikacja dobrego stylu programowania.

Każdy rozdział kończy się pytaniami i zadaniami będącymi, w mojej opinii, cenną pomocą przy samodzielnym studiowaniu podręcznika, tym bardziej, że Dodatek 1 zawiera rozwiązania trudniejszych zadań.

Pozostałe dodatki obejmują:

- listę obiektów predefiniowanych języka Pascal,
- zbiór diagramów składniowych języka (wszystkie w jednym miejscu!),
- pełną definicję składni języka — wyciąg z angielskojęzycznego oryginału normy języka (Autor zatroszczył się o słowniczek).

Książkę uzupełnia spis literatury (każdej pozycji towarzyszy jednozdaniowe objaśnienie zawartości) oraz skorowidz rzeczowy.

Podręcznik przeznaczono, zgodnie ze słowami Autora „...przede wszystkim dla początkujących, a także osób, które zetknęły się już z programowaniem w prostych językach, takich jak BASIC czy Fortran". Ja zaś polecałbym ten podręcznik także tym wszystkim, którzy chcieliby zaznajomić się z zasadami dobrego programowania.

Książkę wydano starannie, szkoda tylko, że na papierze kiepskiej jakości i w dość skromnym nakładzie.

(asd)

KOMPUTER W SZKOLE

ALGORYTM WYZNACZANIA WSPÓŁRZĘDNYCH SŁOŃCA, KSIĘŻYCA I PLANET

(CZĘŚĆ II)

W poprzednim odcinku zamieściliśmy algorytm pozwalający wyznaczyć współrzędne równikowe Słońca i Księżyca. Teraz podamy sposób obliczania tych współrzędnych dla Merkurego, Wenus i Marsa. Kolejne kroki postępowania są podobne do wykonywanych poprzednio. Korzysta się również z tych samych współczynników J1,...,J33. W tabelach 4, 5, 6 zamieszczono elementy pozwalające zbudować kolejne człony wyrażeń DL, DB i DR — odpowiednio dla Merkurego, Wenus i Marsa według schematu podanego dla Słońca.

UWAGA: Obliczając wartości funkcji trygonometrycznych współczynników J1 do J33 należy przedtem pomnożyć je przez wartość 2*PI.

Poprawność konstrukcji tych wyrażeń można sprawdzić porównując wartości liczbowe. Dla daty 28 czerwca 1969 roku (tej samej, dla której wyznaczyliśmy już współczynniki J1,...,J33) powinny one wynosić:

	DL	DB	DR
Merkury	−84564.1	−23166.8	0.37873
Wenus	−860.3	−11478.2	0.72810
Mars	−35479.2	−3900.8	1.46455

Wartości te posłużą do dalszych obliczeń. Warto jednak zaznaczyć, że DR jest odległością planety od Słońca wyrażoną w jednostkach astronomicznych.

Dla każdej z planet należy teraz obliczyć wartość LR, która wynosi:

dla Merkurego LR = (DL/3600)*S + 2*PI*J9 + DLR,
dla Wenus LR = (DR/3600)*S + 2*PI*J12 + DLR,
dla Marsa LR = (DL/3600)*S + 2*PI*J15 + DLR,

gdzie DLR = −17*SIN(J5*2*PI)/3600*S, S podobnie jak poprzednio jest równe PI/180. Należy zwrócić uwagę, by wartość DL obliczona była dla danej planety.

Jeśli LR jest ujemne, to dodajemy do niego 2*PI (LR jest wyrażone w radianach), procedurę powtarzamy aż uzyskamy LR dodatnie.

Dalsze postępowanie, prowadzące do wyznaczenia rektascensji α i deklinacji δ, jest już wspólne dla wszystkich planet. Obliczamy kolejno:

T2 = (JD − 2415020)/36525

$$EP = 23.452294 - 0.0130125*T2 - 1.64*10^{-6}*T2*T2 + 5.03*10^{-7}*T2*T2*T2$$

UWAGA! ERRATA!

1. W poprzednim odcinku wzór na EP zamieściliśmy z dwoma błędami — powyżej jego poprawna postać.
2. Przy obliczaniu wartości funkcji trygonometrycznych należy przedtem współczynniki J pomnożyć przez 2*PI.
3. W wyrażeniu na DL dla Słońca powinno być COS(J8 − J9).

Czytelników przepraszamy za błędy.

DBR = (DB/3600)*S
EPR = EP*S
X1 = DR*COS(DBR)*COS(LR)
X2 = DR*COS(DBR)*SIN(LR)
X3 = DR*SIN(DBR)
X4 = X1 + XS
X5 = X2*COS(EPR) − X3*SIN(EPR) + YS
X6 = X2*SIN(EPR) + X3*COS(EPR) + ZS,

gdzie zamiast XS,YS,ZS należy podstawić obliczone w poprzednim odcinku wartości X4, X5, X6 dla Słońca.

Obliczenie rektascensji α

$\alpha = \mathrm{arctg}(X5/X4)/S$,
gdy $X5 < 0$ i $X4 > 0$ to $\alpha = \alpha + 360$
gdy $X5 < 0$ i $X4 < 0$ to $\alpha = \alpha + 180$
gdy $\alpha < 0$ to $\alpha = \alpha + 180$
$\alpha = \alpha/15$, tj. będzie wyrażone w godzinach i w ułamkach godziny. Ułamek ten możemy zamienić na minuty i sekundy.

Obliczenie deklinacji δ

$\delta = \mathrm{arctg}(X6/(X4*X4 + X5*X5)\uparrow 0.5)/S$,
gdzie symbol ↑ oznacza potęgowanie.
Deklinacja wyrażona jest w stopniach i zawiera się w przedziale $(-90°, +90°)$.

Czytelnik zauważył zapewne, że postępowanie prowadzące do wyznaczenia współrzędnych równikowych planet jest w kilku fragmentach (nie w całości!) identyczne z zamieszczonym w poprzednim numerze. Jednak w celu uzyskania przejrzystości algorytmu przytoczyliśmy go w całości.

Porównanie obliczonych współrzędnych (TEST) z podanymi w Roczniku Astronomicznym (ROCZNIK)

Merkury

	TEST	ROCZNIK
LR/S	341° 15′ 51″	341° 15′ 59″
DBR/S	−6° 26′ 07″	−6° 26′ 17″
DR	0.37873	0.37869
α	$4^h 53^m 45^s$	$4^h 53^m 43^s$
δ	+19° 55′ 16″	+19° 55′ 16″

Wenus

	TEST	ROCZNIK
LR/S	326° 22′ 16″	326° 22′ 05″
DBR/S	−3° 11′ 18″	−3° 11′ 22″
DR	0.72810	0.72808
α	$3^h 16^m 32^s$	$3^h 16^m 30^s$
δ	+15° 05′ 44″	+15° 05′ 48″

Mars

	TEST	ROCZNIK
LR/S	265° 04′ 50″	265° 04′ 54″
DBR/S	−1° 05′ 01″	−1° 04′ 53″
DR	1.46455	1.46463
α	$15^h 58^m 41^s$	$15^h 48^m 43^s$
δ	−23° 43′ 45″	−23° 43′ 07″

Algorytmy, które pozwolą obliczać współrzędne dalszych planet Układu Słonecznego zamieścimy w kolejnych numerach „InforMika". Niezależnie od tego wątku chcielibyśmy zamieszczać również przepisy pozwalające np.: wyznaczać momenty wschodów i zachodów ciał niebieskich, określać ich widoczność itp. Współrzędne stanowią dla nich punkt wyjściowy. Wszystkie te zjawiska rozgrywają się w czasie, dlatego pojęcie czasu jest bardzo istotnym zagadnieniem w astronomii. Poświęcimy mu teraz kilka uwag.

Podstawową jednostką czasu jest doba. Dobą nazywamy odstęp czasu, w którym Ziemia dokonuje jednego pełnego obrotu wokół swojej osi względem pewnego punktu na sferze niebieskiej. Długość doby zależy od tego, względem jakiego punktu określa się okres obrotu Ziemi. Za punkty określające długość doby przyjmuje się:

Tabela 4 (dla Merkurego)

Liczba	T	Funkcja	J5	J7	J8	J13	J15	J16	J17	J19
DL										
38451	0	SIN	0	0	0	0	0	1	0	0
2238	0	SIN	0	0	0	0	0	2	0	0
181	0	SIN	0	0	0	0	0	3	0	0
−52	0	SIN	0	0	0	0	0	0	2	0
37	1	SIN	0	0	0	0	0	1	0	0
−22	0	COS	0	0	0	0	0	1	0	−2
−19	0	SIN	0	0	0	0	0	1	0	−1
17	0	COS	0	0	0	0	0	1	0	−1
17	0	SIN	0	0	0	0	0	4	0	0
−16	0	COS	0	0	0	0	0	2	0	−2
13	0	COS	0	0	1	0	0	−2	0	0
−10	0	SIN	0	0	0	0	0	1	−2	0
−10	0	SIN	0	0	0	0	0	1	2	0
7	0	COS	0	0	1	0	0	−1	0	0
−7	0	COS	0	0	2	0	0	−3	0	0
−5	0	SIN	0	0	0	1	0	−3	0	0
−5	0	SIN	0	0	1	0	0	−1	0	0
−5	0	SIN	0	0	1	0	0	−2	0	0
−4	0	COS	0	0	2	0	0	−4	0	0
4	1	SIN	0	0	0	0	0	2	0	0
4	0	COS	0	0	0	0	0	0	0	1
3	0	COS	0	0	0	1	0	−3	0	0
3	0	SIN	0	0	0	0	0	2	0	−2
DB										
6603	0	SIN	0	0	0	0	0	0	1	0
622	0	SIN	0	0	0	0	0	1	−1	0
615	0	SIN	0	0	0	0	0	1	1	0
64	0	SIN	0	0	0	0	0	2	1	0
DR										
1.53031	0	COS	0	0	0	0	0	0	0	0
−0.14170	0	COS	0	0	0	0	0	1	0	0
−0.00660	0	COS	0	0	0	0	0	2	0	0
−0.00047	0	COS	0	0	0	0	0	3	0	0

Tabela 5 (dla Wenus)

Liczba	T	Funkcja	J1	J5	J7	J8	J12	J13	J14	J16	J19
DL											
2814	0	SIN	0	0	0	0	0	1	0	0	0
−181	0	SIN	0	0	0	0	0	0	2	0	0
−20	1	SIN	0	0	0	0	0	1	0	0	0
12	0	SIN	0	0	0	0	0	2	0	0	0
−10	0	COS	0	0	0	2	0	−2	0	0	0
7	0	COS	0	0	0	3	0	−3	0	0	0
DB											
12215	0	SIN	0	0	0	0	0	0	1	0	0
83	0	SIN	0	0	0	0	0	1	1	0	0
83	0	SIN	0	0	0	0	0	1	−1	0	0
DR											
0.72335	0	COS	0	0	0	0	0	0	0	0	0
−0.00493	0	COS	0	0	0	0	0	1	0	0	0

Tabela 6 (dla Marsa)

Liczba	T	Funkcja	J1	J5	J7	J8	J9	J10	J11	J13	J16	J19
DL												
84378	0	SIN	0	0	0	0	0	1	0	0	0	0
10733	0	SIN	0	0	0	0	0	2	0	0	0	0
1892	0	SIN	0	0	0	0	0	3	0	0	0	0
-646	0	SIN	0	0	0	0	0	0	2	0	0	0
381	0	SIN	0	0	0	0	0	4	0	0	0	0
-306	0	SIN	0	0	0	0	0	1	-2	0	0	0
-274	0	SIN	0	0	0	0	0	1	2	0	0	0
-92	0	SIN	0	0	0	0	0	2	2	0	0	0
83	0	SIN	0	0	0	0	0	5	0	0	0	0
-28	0	SIN	0	0	0	0	0	3	2	0	0	0
25	0	SIN	0	0	0	0	0	2	-2	0	0	0
19	0	SIN	0	0	0	0	0	6	0	0	0	0
-9	0	SIN	0	0	0	0	0	4	2	0	0	0
8	1	SIN	0	0	0	0	0	1	0	0	0	0
7	0	COS	0	0	0	0	0	2	0	-5	0	0
DB												
24314	0	SIN	0	0	0	0	0	0	1	0	0	0
5180	0	SIN	0	0	0	0	0	1	-1	0	0	0
4910	0	SIN	0	0	0	0	0	1	1	0	0	0
1124	0	SIN	0	0	0	0	0	2	1	0	0	0
271	0	SIN	0	0	0	0	0	3	1	0	0	0
132	0	SIN	0	0	0	0	0	2	-1	0	0	0
67	0	SIN	0	0	0	0	0	4	1	0	0	0
18	0	SIN	0	0	0	0	0	2	-1	0	0	0
17	0	SIN	0	0	0	0	0	5	1	0	0	0
-10	0	SIN	0	0	0	0	0	0	3	0	0	0
-9	0	SIN	0	0	0	0	0	1	-2	0	0	0
DR												
0.39528	0	COS	0	0	0	0	0	0	0	0	0	0
-0.07834	0	COS	0	0	0	0	0	1	0	0	0	0
-0.00795	0	COS	0	0	0	0	0	2	0	0	0	0
-0.00121	0	COS	0	0	0	0	0	3	0	0	0	0
-0.00022	0	COS	0	0	0	0	0	4	0	0	0	0

— punkt równonocy wiosennej dla czasu gwiazdowego,
— Słońce „prawdziwe" dla prawdziwego czasu słonecznego,
— Słońce „średnie równikowe" dla czasu średniego słonecznego.

Odstęp czasu między dwoma kolejnymi kulminacjami górnymi (lub dolnymi) punktu równonocy wiosennej nazywamy dobą gwiazdową. Początek doby gwiazdowej na danym południku przypada na moment kulminacji górnej punktu równonocy wiosennej. Czas liczony od początku doby gwiazdowej nosi nazwę czasu gwiazdowego.

Z kolei odstęp czasu między dwoma kolejnymi kulminacjami górnymi (lub dolnymi) środka widomej tarczy Słońca jest prawdziwą dobą słoneczną. Początek prawdziwej doby słonecznej przypada na moment kulminacji dolnej Słońca, a czas liczony od początku prawdziwej doby słonecznej nosi nazwę prawdziwego czasu słonecznego.

Z uwagi na to, że Słońce prawdziwe porusza się po ekliptyce a nie po równiku i w dodatku ze zmienną prędkością, to czas trwania dób prawdziwych słonecznych nie jest stały w ciągu roku; zimą prawdziwa doba słoneczna jest dłuższa niż latem.

Dlatego wprowadza się pojęcie punktu matematycznego poruszającego się ruchem jednostajnym po równiku niebieskim, nazywając go Słońcem średnim. Odstęp czasu między dwoma kolejnymi kulminacjami górnymi (lub dolnymi) średniego Słońca równikowego jest więc średnią dobą słoneczną. Początek średniej doby słonecznej przypada na moment kulminacji dolnej Słońca średniego. Czas liczony od początku średniej doby słonecznej jest więc czasem średnim słonecznym. Różnicę między czasem średnim słonecznym a czasem prawdziwym słonecznym nazywamy równaniem czasu.

Czas gwiazdowy, czas prawdziwy słoneczny i czas średni słoneczny dla danego południka nazywamy odpowiednio miejscowym czasem gwiazdowym, miejscowym czasem prawdziwym słonecznym i miejscowym czasem średnim słonecznym. Miejscowy czas średni słoneczny dla południka Greenwich to czas uniwersalny (UT). Miejscowy czas średni słoneczny dla dowolnego miejsca na powierzchni Ziemi można obliczyć ze wzoru:

$$TM = UT + L,$$

gdzie L jest długością geograficzną danego południka wyrażoną w mierze czasowej i liczoną w kierunku na wschód od południka zerowego przechodzącego przez Greenwich. Czas obowiązujący w danym kraju nazywamy czasem urzędowym. W naszym kraju obowiązują dwa czasy, latem — czas letni (czas wschodnioeuropejski CWE,tj. czas uniwersalny, UT powiększony o 2 godziny), zimą — czas zimowy (czas środkowoeuropejski CSE, tj. czas uniwersalny, UT powiększony o 1 godzinę).

Położenia Słońca, Księżyca i planet obliczane są w oparciu o teorie ich ruchu, gdzie czas jest wielkością niezależną i płynie jednostajnie. Ten czas nazywamy czasem efemerydalnym (ET). Wskutek niejednostajności ruchu wirowego Ziemi występują różnice między czasem efemerydalnym ET a czasem uniwersalnym UT.

$$dT = ET - UT,$$

gdzie dT można obliczyć zgodnie z J. Meeusem wg wzoru:

$$dT = 0.00084*T + 0.000347*T*T,$$

a T oznacza ilość stuleci od roku 1900.
A oto niektóre wartości dT:

Rok	dT
−1000	0.267
−500	0.180
+500	0.056
+1000	0.021
+1500	0.002
+2100	0.003
+2500	0.018
+3000	0.051

(dT wyrażone jest w dobach).
Poprawkę dT można dokładniej obliczać wg L. Schmadela (dokładność 1 sek.), ale za to tylko w przypadku lat 1800—1988. A oto wyrażenie:

$$dt = -0.000014 + 0.001148*T + 0.003357*T^2 - 0.012462*T^3 - 0.022542*T^4 + 0.062971*T^5 + 0.079441*T^6 - 0.146960*T^7 - 0.149279*T^8 + 0.161416*T^9 + 0.145932*T^{10} - 0.067471*T^{11} - 0.058091*T^{12},$$

gdzie dT wyrażone jest w sekundach natomiast T, jak poprzednio, w stuleciach od 1900 r.
W naszych algorytmach przykładowych obliczeń dokonujemy dla 28 czerwca 1969 r. godz. 0 czasu efemerydalnego, czyli dla 27 czerwca 23 godz. 59 min. czasu uniwersalnego, tj. dla 28 czerwca 1969 r. 0 godz. 59 min. czasu środkowoeuropejskiego.

Opracował: IRENEUSZ WŁODARCZYK

ZX-y INACZEJ

JANUSZ MŁODZIANOWSKI TADEUSZ A. ZALESKI

Czy pamiętacie Państwo zamierzchłe czasy ery mikrokomputerowej? Zacznijmy więc od szybkiego przypomnienia. W styczniu 1980 w brytyjskiej firmie Sinclair Research Ltd. opracowano pierwszy popularny mikrokomputer domowy o nazwie ZX80. Pomimo wcześniejszych przypuszczeń o znikomym zbycie ZX80 do końca września tegoż roku sprzedanych zostało ponad 20 000 urządzeń. W marcu 1981 powstał następca ZX80 nazwany ZX81. Mikrokomputer ten rozpoczął istną eksplozję mikroinformatyki stając się w tym czasie najpopularniejszym i najtańszym komputerem świata. Do lutego firma Sinclair wyprodukowała ok. 500 000 urządzeń. W tym samym czasie w USA sprzedawano około 15 000 mikrokomputerów miesięcznie. O tempie rozwoju rynku mikrokomputerowego może świadczyć fakt, że w kwietniu 1982 na Earls Court Computer Fair w Londynie ponad 30 niezależnych firm prezentowało programy i dodatkowy sprzęt do ZX81. Sama firma Sinclaira, licząca początkowo 19 osób, stała się jedną z najbardziej dochodowych w Wielkiej Brytanii. I tak np. rok 1982 zamknął się zyskiem ok. 30 mln funtów, dla porównania rok poprzedni przyniósł tylko 4,6 mln. Rozwijająca się od razu konkurencja firm takich jak: Acorn, Apple, Commodore i innych spowodowała, że Sinclair nie poprzestał na ZX81 ale podjął pracę nad kolejnymi wersjami mikrokomputera, które powszechnie znane są jako seria ZX Spectrum. Niestety, niepowodzenia rynkowe pojazdu elektrycznego C5 oraz kolejnego dziecka Sinclaira, mikrokomputera QL, spowodowały konieczność odprzedania praw autorskich do ZX Spectrum kolejnemu przeciwnikowi — firmie Amstrad. Również w Polsce dały się zauważyć echa rewolucji mikrokomputerowej. Wiele powstających wówczas firm prywatnych i polonijnych zaczęło oferować ZX81, a później ZX Spectrum. Sektor państwowy gospodarki jak gdyby nie zauważył potencjalnego rynku zbytu i bardzo długo zabierał się do rozpoczęcia produkcji. W chwili obecnej na naszym rynku dostępny (?) jest mikrokomputer edukacyjny Elwro 800 Junior. Mikrokomputer ten posiada dwa tryby pracy. Pierwszy tryb zapewnia zgodność programową z systemem ZX. Drugi tryb pracy umożliwia współpracę wielu komputerów w sieci jak też korzystanie z systemu operacyjnego CP/J (zgodnego z CP/M 2.2).

W roku 1981 do akcji wkroczył gigant: amerykańska firma International Business Machines Ltd., czyli IBM, wprowadzając na rynek komputer o nazwie IBM PC. Jak to zwykle bywa z wielkimi koncernami sama nazwa tworzy standard. IBM opublikował wszystkie dane dotyczące komputera. Cały świat rzucił się na produkcję maszyn „zgodnych z IBM". W technice powielania (i sprzedawania) najlepszy okazał się Daleki Wschód. Niewiele czasu upłynęło i sprzęt klasy IBM zaczął docierać do Polski i wypierając urządzenia „niekompatybilne z IBM" stał się nieoficjalnym standardem mikrokomputera. I w ten to sposób doszliśmy do czasów współczesnych. Era Sinclaira należy do archeologów. Ale czy tak musi być? Pamiętajmy, że w Polsce, tylko w rękach prywatnych znajduje się kilkaset tysięcy mikrokomputerów ZX! Nie zapominajmy o nich! Nie każdego bowiem stać na podążanie za trendami światowymi (i modą!) i zakup „Big Blue", którego cena nierzadko jest wyższa od ceny dobrego samochodu. Ponadto, jak mówi stare przysłowie programistów, komputer jest tyle wart, ile osoba go używająca. Nieprawdą jest że Spectrum, ZX81 czy Junior nie nadają się do niczego. Nie zawsze bowiem należy stosować armatę do przybijania gwoździ... W kolejnych odcinkach pragniemy pokazać nowe spojrzenie na mikrokomputery ZX, mamy również nadzieję, że przekonamy Czytelników, iż warto jednak wyciągnąć z szafy starego, zakurzonego ZX i zaprząc go do pracy. Zanim jednak przejdziemy do konkretnego przedstawiania, co i jak można zrobić z mikrokomputerami ZX, pragniemy przestrzec wszystkich „majsterkowiczów". Ingerencja „z lutownicą" w jakikolwiek układ elektroniczny, a zwłaszcza mikroprocesorowy, jest zawsze związana z możliwością jego uszkodzenia. Użytkownikowi nie mającemu wprawy i dopiero początkującemu w technice cyfrowej odradzamy eksperymenty!!! Utrata sprzętu mikrokomputerowego, nawet ZX81, jest zawsze nieprzyjemna i kosztowna.

Proponowane przez nas udoskonalenia ZX-ów można podzielić na kilka grup:
- usprawnienia konstrukcji komputera (dotyczy głównie ZX81),
- budowa prostych interfejsów,
- tworzenie oprogramowania,
- podłączanie urządzeń wejścia/wyjścia do mikrokomputera.

Oczywiście przedstawiane rozwiązania i konstrukcje nie są jedynymi z możliwych i użytkownik po zapoznaniu się z proponowanymi pomysłami może dojść do wniosku, że do jego konkretnych celów właściwa byłaby inna wersja udoskonalenia.

CHARAKTERYSTYKA MIKROKOMPUTERÓW ZX

Mikrokomputery ZX odznaczają się wielką prostotą konstrukcji i właśnie ta cecha stanowi ich ogromną zaletę. Urządzenia te można traktować jako uniwersalne mikrokomputery jednopłytkowe. Po kilku prostych przeróbkach sprzętowych i odpowiednio oprogramowane nadają się świetnie do rejestrowania danych, np. z woltomierzy, do sterowania urządzeń oraz wszędzie tam, gdzie są wykonywane proste czynności lub obliczenia.

W skład systemu mikrokomputerowego ZX wchodzą:
- mikrokomputer z klawiaturą,
- telewizor,
- dowolny magnetofon działający jako pamięć,
- minidrukarka (lub dowolna drukarka z interfejsem),
- zasilacz sieciowy.

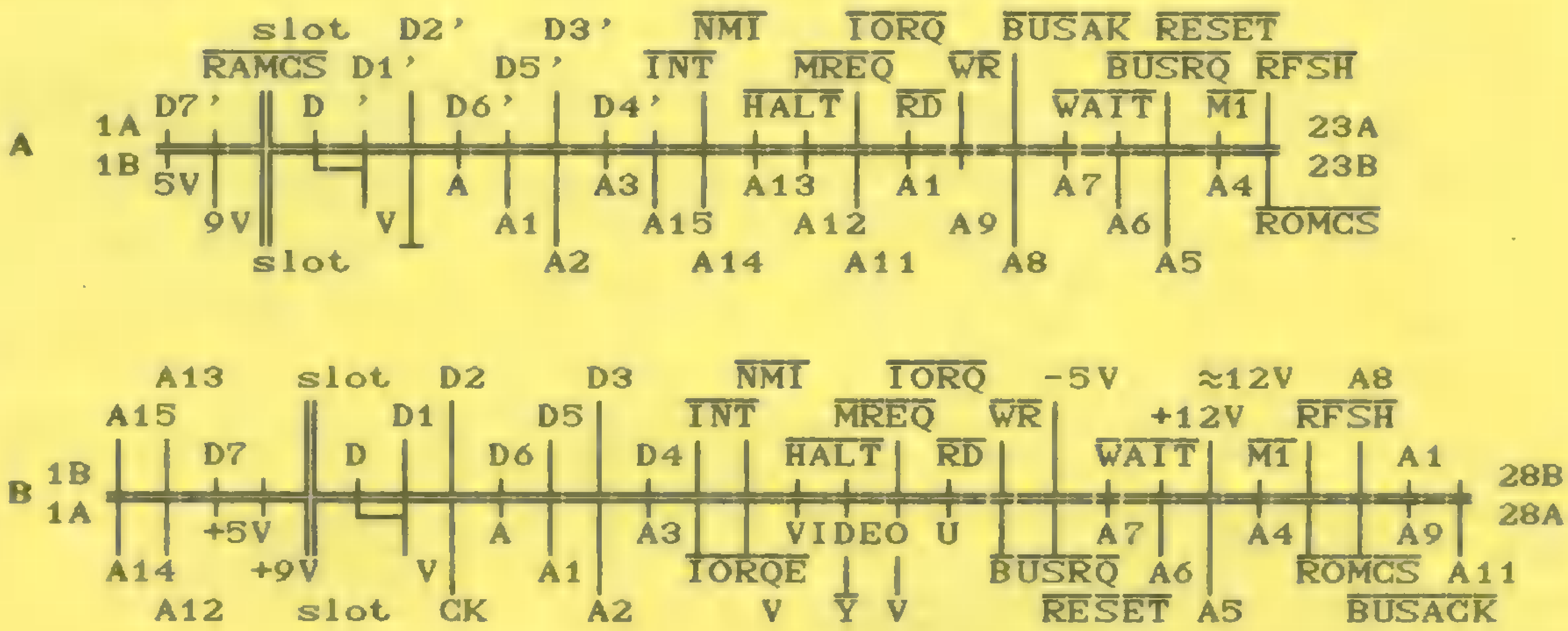
Sam mikrokomputer ZX składa się z kilku bloków funkcjonalnych:
- mikroprocesora,
- pamięci stałej ROM,
- pamięci operacyjnej RAM,
- układów towarzyszących: modulatora TV, stabilizatora i układu logicznego ULA.

Jednostką centralną mikrokomputerów ZX jest mikroprocesor Z-80A. Procesor wykonując ściśle określone instrukcje stanowiące tzw. system operacyjny umożliwia komunikowanie się z operatorem i wykonywanie napisanych przez niego, zwykle w języku wysokiego poziomu, programów.

Generator podstawy czasu mikrokomputera ZX81 jest stabilizowany rezonatorem ceramicznym 6,5 MHz. Generowany przez niego przebieg, po podzieleniu przez dwa (3,25 MHz), jest dostarczony do wejścia CLK procesora. Zegar ZX Spectrum o częstotliwości 14 MHz jest stabilizowany rezonatorem kwarcowym. Po podzieleniu przez 2, częstotliwość 7 MHz steruje układem wizji. Kolejny podział przez 2 (3,5 MHz) daje właściwy takt procesora. Przebieg ten jest okresowo wstrzymywany przez układ ULA.

Mikrokomputer ZX81 jest wyposażony w pamięć ROM o pojemności 8 KB, w której jest zapisany program działania i interpreter języka BASIC. ROM rezyduje w adresach 0000h...1FFFh (XXXXh oznacza liczbę zapisaną w postaci szesnastkowej). Statyczna pamięć operacyjna standardowo o pojemności 1 KB (wersja amerykańska Timex 1000 — 2 KB) zajmuje obszar od 4000h i może być rozszerzona do 56 KB przez dołączenie zewnętrznej pamięci — najczęściej dynamicznej.

Pamięć stała ZX Spectrum ma pojemność 16 KB i zajmuje obszar 0000h...3FFFh. Minimalna pamięć RAM o pojemności 16 KB (ZX Spectrum 16K) może być wewnętrznie rozbudowana do 48



Rys. 1. Złącza systemowe mikrokomputerów ZX81 (A) i ZX Spectrum (B).

KB (ZX Spectrum 48K i ZX Spectrum +). Wersja ZX Spectrum 128K ma stronicowaną pamięć o łącznej pojemności 128 KB.

Całość systemu ZX uzupełnia wysoko specjalizowany układ logiczny ULA, który umożliwia komunikację z urządzeniami wejścia/wyjścia oraz zapewnia poprawną pracę całego systemu. Układ ULA odpowiada funkcjonalnie ok. 20 układom scalonym, w które był wyposażony ZX80. Układ ULA mikrokomputera ZX Spectrum jest bardziej rozbudowany. Zawiera m.in. układ sterujący wyświetlaniem obrazu TV i generator dźwięku. Pierwsze wersje układu ULA (*issue 1*) były wykonane z błędem, który był korygowany dodatkowym układem 74LS00. Wersje kolejne wymagały dodatkowego zewnętrznego tranzystora. Wszystkie sygnały sterujące pracą systemu ZX są wyprowadzone na złącze krawędziowe dostępne dla użytkownika. Rozmieszczenie poszczególnych sygnałów na złączu systemowym ZX81 i ZX Spectrum przedstawione jest na rys. 1. Większość linii złącza systemowego stanowią sygnały kontrolne procesora i tak:

A0..A15 (ZX 81, ZX Spectrum): wyjście, 16-bitowa szyna adresowa mikroprocesora Z80. Pozwala adresować bezpośrednio 65 536 komórek pamięci.

D0..D7 (ZX81, ZX Spectrum): 8-bitowa dwukierunkowa szyna danych procesora Z80. Szyna danych służy do przesyłania informacji pomiędzy poszczególnymi układami mikrokomputera.

/RD (ZX81, ZX Spectrum): wyjście. Sygnał odczytu informacji z pamięci bądź układów wejścia/wyjścia mikrokomputera.

/WR (ZX81, ZX Spectrum): wyjście. Sygnał zapisu informacji do pamięci bądź układów wejścia/wyjścia mikrokomputera.

/MREQ (ZX81, ZX Spectrum): wyjście. Sygnał informujący o odwołaniu się procesora do pamięci.

/IORQ (ZX81, ZX Spectrum): wyjście. Sygnał informujący o odwołaniu się procesora do układów wejścia/wyjścia.

/INT (ZX81, ZX Spectrum): wejście służące do zgłoszenia przerwania maskowalnego.

/NMI (ZX81, ZX Spectrum): wejście służące do zgłoszenia przerwania niemaskowalnego.

/BUSRQ (ZX81, ZX Spectrum): wejście. Sygnał żądania dostępu do szyny procesora. W systemie ZX sygnał ten nie jest wykorzystany i jest podłączony do +5 V przez rezystor 1 kΩ.

/BUSACK (ZX81, ZX Spectrum): wyjście. Sygnał zezwolenia na dostęp do szyny danych. W systemie ZX sygnał ten nie jest wykorzystany.

/WAIT (ZX81, ZX Spectrum): wejście. Sygnał czasowego wstrzymania pracy procesora. W ZX81 sygnał ten generowany jest pośrednio z ULA.

/HALT (ZX81, ZX Spectrum): wyjście. Sygnał informujący o wykonaniu instrukcji HALT i przejściu do stanu wstrzymania.

/M1 (ZX81, ZX Spectrum): wyjście. Sygnał informujący o wykonywaniu przez procesor cyklu pobrania instrukcji (**fetch**).

RFSH (ZX81, ZX Spectrum): wyjście. Sygnał informujący o wykonywaniu cyklu odświeżania pamięci dynamicznych.

Φ0 (ZX81, ZX Spectrum): wyjście, zegar procesora.

RESET (ZX81, ZX Spectrum): wejście. Sygnał wymuszający zerowanie procesora. Generowany automatycznie po włączeniu zasilania.

Pozostałe sygnały szyny systemowej systemu ZX obsługują pamięć oraz układ generacji obrazu:

/ROMCS (ZX81, ZX Spectrum): wejście. Sygnał wyłączający pamięć systemową ROM.

/RAMCS (ZX81, —): sygnał wyłączający wewnętrzny 1 KB pamięci RAM.

IORQGE (—, ZX Spectrum): wejście IORQ układu logicznego ULA; sygnał ten jest przyłączony do sygnału IORQ procesora przez szeregowy rezystor.

U (—, ZX Spectrum): różnicowy sygnał chrominancji.
V (—, ZX Spectrum): różnicowy sygnał chrominancji.
Y (—, ZX Spectrum): zespolony sygnał luminancji i synchronizacji.

Mikrokomputery ZX81 i ZX Spectrum wyposażone są w klawiaturę membranową, składającą się z 40 pól kontaktowych. W odróżnieniu od płaskiej klawiatury ZX81, pokrycie klawiatury ZX Spectrum

wykonane jest z wytłoczki gumowej, która w działaniu sprawia wrażenie normalnej, tj. ruchomej klawiatury. Wersja ZX Spectrum+ i kolejne wyposażone są w klawiaturę, której klawisze wykonane są z plastyku.

Mikrokomputery ZX współpracują z ekranem telewizyjnym. Wbudowany modulator UHF UM1223 wytwarza sygnał wizyjny w kanale 36, tj. 591,5 MHz, który doprowadza się do telewizora poprzez gniazdo antenowe. ZX81 wyświetla obraz czarno-biały o rozdzielczości 32 na 22 znaków. ZX Spectrum ma dodatkowo możliwość generowania obrazu kolorowego w systemie PAL i grafiki o rozdzielczości 256 na 176 punktów. W ZX81 obraz jest generowany przez program zapisany w pamięci ROM, podczas gdy w ZX Spectrum wyświetlaniem zajmuje się całkowicie układ ULA. W ZX Spectrum sygnał wizyjny PAL generowany jest w koderze LM 1889N.

Zewnętrzną pamięć masową mikrokomputerów ZX stanowić może dowolny magnetofon kasetowy. Do współpracy z nim służą dwa gniazda oznaczone odpowiednio MIC i EAR. Nowsze wersje mikrokomputera ZX Spectrum, produkowane już przez firmę Amstrad, mają wewnętrznie wmontowany magnetofon kasetowy (ZX Spectrum+2) lub 3" napęd dysków elastycznych (ZX Spectrum+3). Spotykane są również zewnętrzne stacje dysków 5,25" i 3" (np. spotykana w Polsce stacja TIMEX).

Mikrokomputer ZX może być uzupełniony drukarką. W zależności od jej typu może ona wymagać stosowania specjalizowanego interfejsu i oprogramowania.

Na płytce komputera znajduje się również stabilizator napięcia +5 V. Przetwornica mikrokomputera ZX Spectrum dodatkowo generuje napięcia +12 V oraz −5 V. Mikrokomputery ZX wymagają zewnętrznego zasilacza niestabilizowanego napięcia stałego +9 V (ZX81 minimum 700 mA, ZX Spectrum minimum 1,2 A).

Komputery ZX mają wbudowane interpreter i edytor języka BASIC. Panuje powszechna opinia, że język ten (sugeruje to nawet nazwa, z ang. *Beginner's All-purpose Symbolic Instruction Code*) jest przeznaczony dla początkujących. Rzeczywiście, opanowanie znaczenia kilkunastu instrukcji prostej wersji języka BASIC nie jest zadaniem trudnym, ale ubogi zestaw instrukcji i wprowadzone uproszczenia istotnie ograniczają klasę algorytmów, które można swobodnie wyrazić w tym języku. Niemniej, praktycznie wszystkie liczące się na rynku mikrokomputery osobiste i profesjonalne, nie wyłączając IBM i HP, zawierają w swej pamięci stałej edytor i interpreter BASIC. Pierwszy umożliwia pisanie programów, drugi — ich wykonywanie.

Język BASIC nie doczekał się oficjalnego raportu, stąd trudno znaleźć wzorzec, w stosunku do którego wersję ZX można ocenić. Porównanie dostępnych w niej instrukcji z wersjami BASIC w innych mikrokomputerach wypada dość pomyślnie, szczególnie dla ZX Spectrum. BASIC ZX81 nie ma jedynie instrukcji DATA, READ, RESTORE.

Systemy ZX doczekały się szerokiej gamy oprogramowania. Oprócz niezliczonej ilości, często bardzo złożonych, gier komputerowych i symulatorów dostępne jest także oprogramowanie użytkowe. Nie wdając się w szczegóły wymienimy tu:

- kompilatory i interpretery języków wyższego rzędu (Pascal, FORTH, Micro Prolog, Logo, rozszerzone wersje BASIC-a),
- asemblery i monitory kodu maszynowego,
- programy graficzne,
- programy wspomagania inżynierskiego (w tym także udane analizatory obwodów elektrycznych).

Oczywiście trzeba sobie zdawać sprawę, że ograniczenia struktury systemu nakładają ograniczenia na poszczególne programy, które są uboższe niż ich odpowiedniki na dużych systemach komputerowych.

W tym odcinku zamieścimy wiadomości wstępne, niezbędne do ogólnego przypomnienia sobie wiadomości o systemach ZX, a mianowicie ogólny opis systemu i dokładny opis zmiennych systemowych.

CHARAKTERYSTYKA SYSTEMU KOMPUTERÓW ZX

W komputerach ZX nie istnieje ścisłe pojęcie systemu operacyjnego mikrokomputera. Najczęściej przyjmuje się, że jest to zbiór programów nadzorujących korzystanie z zasobów sprzętowych, do których zalicza się mikroprocesor, pamięć operacyjną oraz urządzenia wejścia/wyjścia. Akceptacja tej definicji umożliwia zaliczenie do systemu operacyjnego nie tylko programów czuwających nad poprawną pracą mikrokomputera i pośredniczących w komunikacji między operatorem i sprzętem (monitor), ale też program redagujący (edytor) i realizujący (interpreter).

W przypadku mikrokomputerów ZX81 i ZX Spectrum wymienione programy są ze sobą tak ściśle związane, że trudno je (z pewnymi wyjątkami) jednoznacznie rozdzielić. System operacyjny mikrokomputerów ZX oraz interpreter języka BASIC umieszczono w pamięci stałej mikrokomputera (ROM) o łącznej (wraz z tablicami) objętości 8 KB (adresy 0000h...1FFFh) dla ZX81 i 16 KB (adresy 0000h...3FFFh) dla ZX Spectrum.

Zawartość całej pamięci ROM w mikrokomputerach ZX nazywa się programem MONITOR lub ROM PROGRAM.

Poniżej przedstawiamy najistotniejsze procedury MONITORA (w nawiasach podano odpowiednie adresy procedur dla obu mikrokomputerów).

- Procedura inicjująca działanie mikrokomputera (ZX81 03CBh, ZX Spectrum 11CBh) po doprowadzeniu zasilania. Ustala ona rozmiar pamięci dostępnej dla interpretera BASIC, wartości rejestrów I i IY procesora (odpowiednio ZX81 1Eh i 4000h, ZX Spectrum 3Fh i 5C3Ah), wybiera tryb SLOW (ZX81), instaluje minimalny programowy stos maszynowy oraz formuje obszar pamięci operacyjnej, gdzie przechowywana jest informacja o obrazie TV (D__FILE). Wszystkie zmienne systemowe (omówione dalej) otrzymują swoje wartości.
- Procedury redagujące języka BASIC (ZX81 0419h, ZX Spectrum 0F2Ch) umożliwiają pisanie linii programu z kontrolą poprawności składni, ich poprawianie (ruch kursora, edycja wybranej linii) i automatyczny przegląd programu.
- Procedura interpretująca linię programu w języku BASIC (ZX81 0CBAh, ZX Spectrum 1B17h), czyli procedura nadzorująca realizację linii programu.
- Zestaw procedur realizujących instrukcje języka BASIC (ZX81 0DABh, ZX Spectrum 1CEEh).
- Procedura realizująca wyrażenia numeryczne i tekstowe (ZX81 0F55h, ZX Spectrum 24FBh) wraz z interpretacją i obliczaniem wartości funkcji.
- Zestaw procedur używanych głównie przez program kalkulatora zmiennopozycyjnego (ZX81 158Ah, ZX Spectrum 2D4Fh).
- Procedura kalkulatora zmiennopozycyjnego (ZX81 199Dh, ZX Spectrum 335Bh).
- Procedury współpracy z magnetofonem (ZX81 02F6h, ZX Spectrum 04C2h) jako urządzeniem wejścia/wyjścia.
- Procedury nadzorujące obsługę wyświetlania obrazu (ZX81 0207h, w ZX Spectrum obraz jest generowany sprzętowo) wraz z kontrolowaniem stanu klawiatury (ZX81 02BBh, ZX Spectrum 02BFh), a więc również współpraca z urządzeniami wejścia/wyjścia.
- Zestaw najczęściej używanych procedur, które mogą być wywoływane 1--bajtowym rozkazem maszynowym RST n.

Podane procedury są uzupełnione licznymi tablicami, między innymi:

- znaków (ZX81 007Eh, ZX Spectrum 0095h), przypisującą jednoznacznie odpowiednim kodom znaki lub całe słowa;
- instrukcji (ZX81 0C29h, ZX Spectrum 1A48h), umożliwiającą zidentyfikowanie instrukcji w czasie interpretacji linii programu w języku BASIC (instrukcje w linii programu reprezentowane są przez 1 bajt) i określenie sposobu jej wykonania;
- stałych i funkcji (ZX81 1915h, ZX Spectrum 32C5h) używanych przez procedurę kalkulatora zmiennopozycyjnego;
- generatora znaków alfanumerycznych (ZX81 1E00h, ZX Spectrum 3D00h), określającego kształt poszczególnych znaków na ekranie.

Pamięć RAM, znajdująca się pod kontrolą systemu operacyjnego, podzielona jest na kilka obszarów, które przechowują różne rodzaje informacji o stanie mikrokomputera. Różne systemy operacyjne stosują różne sposoby podziału pamięci. Podział ten przedstawia się w postaci tzw. mapy pamięci.

Zwykle programista nie musi orientować się, gdzie, jak i w jaki sposób przechowywana jest informacja. Jednakże znajomość systemu operacyjnego umożliwia świadomą ingerencję w jego działanie. Niekiedy taka ingerencja prowadzi do efektywniejszego wykonywania programów. Ponadto część procedur może być z powodzeniem używanych przez programistów realizujących swe własne programy w kodzie maszynowym. Omówimy niektóre z nich. Czytelników nie znających zasad programowania w kodzie maszynowym i związanych z tym nierozerwalnie mnemonicznych skrótów rozkazów mikroprocesora Z80 odsyłamy do bogatej literatury, również w języku polskim. Zasady pisania

programów maszynowych dla mikrokomputerów ZX przedstawione zostaną w kolejnych częściach cyklu.

ZMIENNE SYSTEMOWE MIKROKOMPUTERA ZX81

System operacyjny wymaga pewnego obszaru pamięci, gdzie byłyby przechowywane charakterystyczne parametry umożliwiające kontrolowanie poprawnej pracy całego mikrokomputera, tak też jest w przypadku mikrokomputerów ZX81 i ZX Spectrum. Obszar ten nazwano obszarem zmiennych systemowych (SYS_VAR). Zawarte w nich informacje umożliwiają ocenę stanu mikrokomputera i zmienianie go z wnętrza procedur realizowanych w kodzie maszynowym, jak również z poziomu języka BASIC. Poniżej zestawiono zmienne systemowe (w nawiasach kolejno podano: liczbę bajtów, adresy w kodzie szesnastkowym i dziesiętnym).

ERR_NR (1, 4000h, 16384). Kod raportu o błędzie zmniejszony o 1. Umieszczenie w tej zmiennej wartości mniejszej niż 128 powoduje zatrzymanie pracy wykonywanego programu (por. tabl. 1) i zgłoszenie:

- standardowego komunikatu o błędzie (0...14),
- niestandardowego komunikatu o błędzie (15...35 i 98...127) oraz podanie numeru linii, w trakcie realizacji której system wykrył wystąpienie wskazanej wartości zmiennej ERR_NR, lub:
- niezgłoszenie komunikatu i/lub dezorganizację (tzw. zawieszenie) pracy systemu (36...97).

FLAGS (1, 4001h, 16385). Zespół bitów kontrolnych systemu.

Bit 0 — wskaźnik druku dodatkowej spacji:
 1 — spację zignorować,
 0 — umieścić dodatkową spację poprzedzającą lub kończącą druk nazwy instrukcji (słowa).
Bit 1 — wskaźnik kontrolny drukarki:
 1 — wyprowadzenie znaku na drukarkę,
 0 — wyprowadzenie znaku na ekran.
Bit 2 — wskaźnik wyboru trybu pracy klawiatury:
 1 — tryb inny niż K (z ang. *keyword*),
 0 — tryb K.
Bit 6 — wskaźnik typu zmiennej umieszczonej ostatnio na szczycie stosu kalkulatora:
 1 — liczba,
 0 — parametry łańcucha.
Bit 7 — wskaźnik rodzaju pracy:
 1 — interpretacja linii programu w języku BASIC,
 0 — sprawdzenie poprawności składni linii.

ERR_SP (2, 4002h, 16386). Adres pierwszej pozycji (dna) stosu maszynowego; równocześnie wskaźnik stosu GOSUB przechowującego powrotne numery linii z podprogramów, jeżeli:
errsp = PEEK 16386 + 256*PEEK 16387,
to wartość:
PEEK (errsp + 2) + 256*PEEK (errsp + 3)
wskazuje numer wiersza (15872 oznacza pusty stos GOSUB), do którego nastąpi skok przy wykonaniu instrukcji RETURN (jest to zwiększony o 1 numer wiersza, w którym wystąpiła instrukcja GOSUB).
RAMTOP (2, 4004h, 16388). Adres pierwszej komórki pamięci powyżej obszaru kontrolowanego przez interpreter języka BASIC. Zmniejszenie tej wartości i przeniesienie stosu maszynowego i stosu GOSUB (np. przez NEW) umożliwia umieszczenie powyżej adresu wyznaczonego przez RAMTOP programu w kodzie maszynowym lub danych.
MODE (1, 4006h, 16390). Wskaźnik wyboru trybu pracy klawiatury:
 0 — tryb L (z ang. *letter*),
 1 — tryb F (z ang. *function*),
 4 — tryb G (z ang. *graphics*).
PCC (2, 4007h, 16391). Numer interpretowanej linii, zmienna ta zostaje podana przy raporcie o błędzie (po znaku „/").
VERSN (1, 4009h, 16393). Oznaczenie początku bloku pamięci RAM, od którego są pobierane zmienne przez instrukcję SAVE (zapis na magnetofon).
E_PPC (2, 400Ah, 16394). Numer linii programu z kursorem (linia o tym numerze ukaże się w dolnej części ekranu po naciśnięciu EDIT).
D_FILE (2, 400Ch, 16396). Adres pierwszej komórki obszaru pamięci używanego do zapamiętania obrazu TV (zwanego obszarem D_FILE). Obszar ten znajduje się zawsze bezpośrednio za programem w języku Basic. Stąd wartość wyrażenia
PEEK 16396 + 256*PEEK 16397 − 16509
określa rozmiar programu w bajtach.
DF_CC (2, 400Eh, 16398). Adres w obszarze D_FILE, pod którym będzie umieszczony kolejny znak przeznaczony do wydruku.
VARS (2, 4010h, 16400). Adres pierwszej komórki obszaru pamięci przechowującego zadeklarowane zmienne (zwany obszarem VARS), niech:
vars = PEEK 16400 + 256*PEEK 16401.
DEST (2, 4012h, 16402). Adres pierwszej litery nazwy poszukiwanej zmiennej w linii programu, gdy jest to zmienna prosta użyta pierwszy raz, lub w obszarze VARS, gdy występowała wcześniej lub jest to zmienna indeksowana.
E_LINE (2, 4014h, 16404). Adres obszaru roboczego edytora, gdzie jest umieszczana redagowana linia programu (lub zlecenie bezpośrednie), bądź kopia linii programu po EDIT. Równocześnie zmienna ta określa koniec obszaru zmiennych, jeżeli
eline = PEEK 16404 + 256*PEEK 16405,
to
eline − vars
określa wielkość obszaru zmiennych, a
eline − 16393
rozmiar pamięci, jaki zostanie zapamiętany na taśmie magnetofonowej przez instrukcję SAVE, gdyż ostatnim zapisywanym bajtem jest bajt o adresie eline − 1
CH_ADD (2, 4016h, 16406). Adres znaku (kodu instrukcji), który będzie interpretowany w linii programu lub w zleceniu bezpośrednim.
X_PTR (2, 4018h, 16408). Adres znaku w linii programu lub w zleceniu bezpośrednim (w obszarze roboczym interpretera), który jest poprzedzony kursorem informującym o błędzie składni w linii.
STKBOT (2, 401Ah, 16410). Adres początku stosu kalkulatora zmiennopozycyjnego.
STKEND (2, 401Ch, 16412). Adres końca stosu kalkulatora zmiennopozycyjnego.
BREG (1, 401Eh, 16414). Rejestr roboczy B używany głównie przez kalkulator zmiennopozycyjny.
MEM (2, 401Fh, 16415). Adres obszaru roboczego procedury kalkulatora.
DF_SZ (1, 4022h, 16418). Liczba linii w dolnej części ekranu (standardowo 2) zarezerwowanych dla systemu. Zmniejszenie tej liczby umożliwia umieszczenie tam własnych wydruków (przed wykonaniem instrukcji SCROLL liczba ta musi być większa niż 0, przed INPUT większa niż 1).
S_TOP (2, 4023h, 16419). Numer linii, od której będzie emitowany tekst programu na ekranie TV.
LAST_K (2, 4025h, 16421). Odczyt stanu klawiatury.
DEBOUNCE (1, 4027h, 16423). Zmienna informująca czy nastąpiła zmiana w wyborze klawisza.
MARGIN (1, 4028h, 16424). Stała zależna od systemu TV.
NXTLN (2, 4029h, 16425). Adres początku kolejnej linii programu, która będzie interpretowana. Zmienna ta umożliwia umieszczenie procedur w kodzie maszynowym w liniach REM w dowolnym miejscu programu w języku BASIC, jeżeli są to procedury przemieszczalne (relokowalne), czyli niezależne od adresu, pod którym się znajdują, np.:

```
9997 RAND USR (5+PEEK 16425+
     +256*PEEK 16426)
9998 REM procedura w kodzie maszy-
     nowym
9999 RETURN
```

OLDPPC (2, 402Bh, 16427). Numer linii, do której nastąpi skok po wykonaniu instrukcji CONT.
FLAGX (1, 402Dh, 16429). Bity kontrolne systemu.

Bit 0 — wskaźnik typu zmiennej:
 1 — zmienna prosta,
 0 — zmienna indeksowana.

Bit 1 — wskaźnik obecności zmiennej w obszarze VARS:
 1 — zmienna nie występowała,
 0 — zmienna znajduje się w obszarze VARS.

Bit 5 — wskaźnik trybu pracy:
 1 — tryb INPUT,
 0 — tryb EDIT.

Bit 6 — wskaźnik oczekiwania w trybie INPUT na:
 1 — wyrażenie arytmetyczne,
 0 — wyrażenie tekstowe.

STRLEN (2, 402Eh, 16430). Długość zidentyfikowanej zmiennej alfanumerycznej lub linii programu.
T_ADDR (2, 4030h, 16432). Adres następnego elementu w tablicy parametrów (określającej składnię realizowanej instrukcji), który będzie pobrany do interpretacji.
SEED (2, 4032h, 16434). Podstawa generatora liczb pseudolosowych. Wartość argumentu instrukcji RAND (dla 0 jest to liczba losowa) lub wartość ostatnio użytej funkcji RND pomnożona przez 65536.
FRAMES (2, 4034h, 16436). Licznik obrazów TV. Zawartość tej zmiennej jest zmniejszana co 0,02 s, tak, by bit 15 równał się 1. Zmienna ta jest stosowana przez instrukcję PAUSE, która umieszcza tu swój argument (bit 15 = 0). Zmienna

FRAMES umożliwia pomiar czasu (w trybie SLOW) do ponad 10 minut. Np. przed pomiarem

POKE 16437,255
POKE 16436,255

a na zakończenie pomiaru

time = 65535 − PEEK 16436 − 256*PEEK 16437

wartość time/50 będzie odpowiadać czasowi pomiaru w sekundach.

COORDS (2, 4036h, 16438). Zmienna przechowuje argumenty ostatnio użytej instrukcji PLOT lub UNPLOT.

PR__CC (1, 4038h, 16440). Mniej znacząca część adresu, pod którym zostanie umieszczony kod znaku przeznaczonego do wydrukowania na drukarce.

S__POSN (2, 4039h, 16441). Współrzędne następnego wydruku na ekranie TV:

24 − PEEK 16442
podaje numer wiersza,
33 − PEEK 16441
podaje numer kolumny.

CDFLAG (1, 403Bh, 16443). Bity kontrolne systemu.

Bit 0 — wskaźnik stanu klawiatury:
 1 — klawisz został wciśnięty,
 0 — klawisz nie został wciśnięty.
Bit 6 — zasadniczy wskaźnik trybu SLOW/FAST:
 1 — tryb SLOW,
 0 — tryb FAST.
Bit 7 — pomocniczy wskaźnik trybu SLOW/FAST, wymuszający tryb SLOW przy obowiązującym trybie FAST (np. przy INPUT):
 1 — tryb SLOW,
 0 — tryb zgodny ze stanem określonym przez bit 6.

PRBUFF (33, 403Ch, 16444). Bufor drukarki.

MEMBOT (30, 405Dh, 16477). Obszar pamięci roboczej kalkulatora zmiennopozycyjnego. Może przechowywać sześć liczb w postaci zmiennopozycyjnej.

ZMIENNE SYSTEMOWE MIKROKOMPUTERA ZX SPECTRUM

KSTATE (8, 5C00h, 23552). Zdublowany system (zestaw pierwszy KSTATE__0...KSTATE__3 i drugi KSTATE__4...KSTATE__7) zmiennych określających stan klawiatury. W zmiennej KSTATE__4 znajduje się kod główny wciśniętego klawisza (FFh, gdy klawiatura nie jest używana albo wciśnięty jest tylko CAPS__SHIFT lub tylko SYMBOL__SHIFT). W KSTATE__7 znajduje się właściwy kod ASCII (po uwzględnieniu równoczesnego wciśnięcia CAPS__SHIFT lub SYMBOL__SHIFT) ostatnio wybranego klawisza. KSTATE__5 (powtarzanie) i KSTATE__6 (opóźnienie) służą określaniu relacji czasowych przy automatycznym powtarzaniu odczytu. Jeżeli dwa różne klawisze zostaną kolejno wciśnięte w odstępie krótszym niż okres automatycznego powtarzania (z ang. *autorepeat*, standardowo 0,1 s), to analogiczne dane o drugim klawiszu są umieszczane w zestawie pierwszym (KSTATE__0...KSTATE__3).

LAST__K (1, 5C08h, 23560). Właściwy kod ASCII ostatnio wybranego klawisza.

REPDEL (1, 5C09h, 23561). Opóźnienie, czas (wielokrotność 1/50 s), jaki musi upłynąć przed rozpoczęciem realizacji automatycznego powtarzania odczytu klawisza (standardowo 23h, czyli 0,7 s).

REPPER (1, 5C0Ah, 23562). Powtarzanie, czas (wielokrotność 1/50 s) między kolejnymi automatycznymi odczytami klawisza (standardowo 05h, czyli 0,1 s).

DEFADD (2, 5C0Bh, 23563). Adres argumentu występującego w deklaracji DEF FN).

K__DATA (1, 5C0Dh, 23565). Informacja o interpretacji kodu FLASH, BRIGHT lub INVERSE:

 1 — włączyć (ON),
 0 — wyłączyć (OFF).

TVDATA (2, 5C0Eh, 23566). Przechowuje kod użytego znaku kontrolnego INK...OVER (w TVDATA__lo, czyli w mniej znaczącym bajcie) i jego argument (w TVDATA__hi, czyli w bardziej znaczącym bajcie). Dla znaków kontrolnych AT i TAB w TVDATA__lo jest przechowywany jego kod, a w TVDATA__hi pierwszy argument (drugi jest zapamiętywany chwilowo w rejestrze A procesora).

STRMS (38, 5C10h, 23568). Adresy względne (z ang. *offsets*) kanałów ustalających komunikację procesora z urządzeniami wejścia/wyjścia.

CHARS (2, 5C36h, 23606). Adres początku generatora znaków zmniejszony o 0100h. Zawiera kolejno wzorce znaków o kodach 20h...7Fh. Można ustalić własny generator tych znaków w pamięci RAM (umieszczając analogicznie przekształcony adres jego początku w CHARS).

RASP (1, 5C38h, 23608). Czas (wielokrotność 1/50 s) trwania sygnału ostrzegawczego (standardowo 40h, czyli 1,28 s).

PIP (1, 5C39h, 23609). Czas (wielokrotność 1/50 s) trwania sygnału po naciśnięciu klawisza (standardowo 00h).

ERR__NR (1, 5C3Ah, 23610). Kod raportu o błędzie zmniejszony o 1. Umieszczenie w tej zmiennej dowolnej wartości i wywołanie

USR 88

spowoduje zatrzymanie pracy wykonywanego programu i zgłoszenie

— standardowego komunikatu o błędzie (0...28) lub zakończenie pracy (255),
— niestandardowego komunikatu o błędzie (29...254).

Oprócz symbolu, numeru wiersza i pozycji w wierszu ukazuje się również słowny komentarz.

FLAGS (1, 5C3Bh, 23611). Zespół bitów kontrolnych systemu:

Bit 0 — wskaźnik druku dodatkowego odstępu (spacji):
 1 — spację pominąć,
 0 — umieścić dodatkową spację poprzedzającą lub kończącą druk nazwy instrukcji (słowa).
Bit 1 — wskaźnik kontrolny drukarki:
 1 — wyprowadzenie znaku na drukarkę,
 0 — wyprowdzenie znaku na ekran.
Bit 2 — wskaźnik trybu, w którym będzie drukowany kolejny znak:
 1 — w trybie L (z ang. *letter*) lub C (z ang. *capital*),
 0 — w trybie K (z ang. *keyword*).
Bit 3 — wskaźnik wyboru modu klawiatury:
 1 — tryb L lub C,
 0 — tryb K.
Bit 4 — wskaźnik wyboru kanału wejścia/wyjścia:
 1 — kanał K (z ang. *keyboard*),
 0 — kanał inny niż K.
Bit 5 — wskaźnik stanu klawiatury:
 1 — wciśnięty nowy klawisz,
 0 — klawiatura nie używana, gotowa do odczytu.
Bit 6 — wskaźnik typu zmiennej umieszczonej ostatnio na szczycie stosu kalkulatora:
 1 — liczba,
 0 — parametry łańcucha.
Bit 7 — wskaźnik rodzaju pracy:
 1 — interpretacja linii programu,
 0 — sprawdzanie poprawności składni linii.

TV__FLAG (1, 5C3Ch, 23612). Zespół bitów kontrolnych systemu:

Bit 0 — wskaźnik wyboru obszaru ekranu:
 1 — wybór dolnej (systemowej) części,
 0 — wybór górnej (głównej) części.
Bit 3 — wskaźnik zmiany trybu EDIT/INPUT:
 1 — nastąpiła zmiana,
 0 — bez zmian.
Bit 4 — wskaźnik automatycznego wyświetlania tekstu programu (z ang. *listing*):
 1 — wyświetlać,
 0 — nie wyświetlać.
Bit 5 — wskaźnik stanu systemowej części ekranu:
 1 — oczyścić po naciśnięciu klawisza,
 0 — pozostawić bez zmian.

ERR__SP (2, 5C3Dh, 23613). Adres pierwszej pozycji (dna) stosu maszynowego, równocześnie wskaźnik stosu GOSUB (por. ze zmienną systemową ZX81 o tej samej nazwie).

LIST__SP (2, 5C3Fh, 23615). Adres (w stosie maszynowym), pod którym znajduje się adres powrotu z procedury realizującej automatyczne wyświetlanie tekstu programu.

MODE (1, 5C41h, 23617). Wskaźnik wyboru trybu pracy klawiatury:

 0 — tryb K (z ang. *keyword*), L (z ang. *lower*) lub C (z ang. *capital*),
 1 — tryb E (z ang. *extend*),
 2 — tryb G (z ang. *graphics*).

NEWPPC (2, 5C42h, 23618). Numer następnej linii do interpretacji.

NSPPC (1, 5C44h, 23620). Numer instrukcji w linii, która będzie interpretowana.

PPC (2, 5C45h, 23621). Tak jak dla ZX81.

SUBPPC (1, 5C47h, 23623). Numer instrukcji w linii, która jest interpretowana.

BORDCR (1, 5C48h, 23624). Kod koloru ramki (BORDER) i systemowej części obrazu TV (PAPER) pomnożony przez 8, plus kod koloru znaków (INK) dla tej części ekranu.

E__PPC (2, 5C49h, 23625). Tak jak dla ZX81.

VARS (2, 5C4Bh, 23627). Adres pierwszej komórki obszaru pamięci przechowującego zadeklarowane zmienne (zwanego obszarem VARS), niech

vars = PEEK 23627 + 256*PEEK 23628

DEST (2, 5C4Dh, 23629). Tak jak dla ZX81.

CHANS (2, 5C4Fh, 23631). Podstawa

adresu, pod którym znajdują się informacje o kanałach ustalających komunikację procesora z urządzeniami wejścia/wyjścia.

CURCHL (2, 5C51h, 23633). Adres, pod którym znajduje się informacja o obowiązującym (wybranym) kanale.

PROG (2, 5C53h, 23635). Adres, od którego jest przechowywany w pamięci tekst programu w języku BASIC (obszar PROG), jeżeli

prog = PEEK 23635 + 256*PEEK 23636

to

vars – prog

określa rozmiar programu w bajtach.

NXTLIN (2, 5C55h, 23637). Adres początku kolejnej linii programu.

DATADD (2, 5C57h, 23639). Adres (w obszarze PROG) wskazjący separator kończący ostatnio wczytany element z instrukcji DATA.

E__LINE (2, 5C59h, 23641). Tak jak dla ZX81.

K__CUR (2, 5C5Bh, 23643). Adres kursora.

CH__ADD (2, 5C5Dh, 23645). Tak jak dla ZX81.

X__PTR (2, 5C5Fh, 23647). Tak jak dla ZX81.

WORKSP (2, 5C61h, 23649). Adres obszaru roboczego interpretera. Są tam chwilowo umieszczane dane przy instrukcji BASIC INPUT.

STKBOT (2, 5C63h, 23651). Tak jak dla ZX81.

STKEND (2, 5C65h, 23653). Tak jak dla ZX81.

BREG (1, 5C67h, 23655). Tak jak BERG dla ZX81.

MEM (2, 5C68h, 23656). Tak jak dla ZX81.

FLAGS2 (1, 5C6Ah, 23658). Zespół bitów kontrolnych systemu.

Bit 0 — wskaźnik stanu ekranu:
- 1 — obszar ekranu nie jest pusty,
- 0 — obszar ekranu jest pusty.

Bit 1 — wskaźnik stanu bufora drukarki:
- 1 — bufor drukarki nie jest pusty,
- 0 — bufor drukarki jest pusty.

Bit 2 — wskaźnik cudzysłowu:
- 1 — analizowany znak jest elementem łańcucha,
- 0 — analizowany znak nie jest elementem łańcucha.

Bit 3 — wskaźnik CAPS__LOCK:
- 1 — drukować w trybie C (z ang. *capital*),
- 0 — drukować w trybie L (z ang. *lower*).

Bit 4 — wskaźnik obowiązującego kanału wejścia/wyjścia.
- 1 — kanał K (z ang. *keyboard*),
- 0 — kanał inny niż K.

DF__SZ (1, 5C6Bh, 23659). Tak jak dla ZX81.

S__TOP (2, 5C6Ch, 23660). Tak jak dla ZX81.

OLDPPC (2, 5C6Eh, 23662). Numer linii, do której nastąpi skok po wykonaniu instrukcji CONTINUE.

OSPPC (1, 5C70h, 23664). Numer instrukcji w linii, która będzie interpretowana przy zleceniu CONTINUE.

FLAGX (1, 5C71h, 23665). Zespół bitów kontrolnych systemu.

Bit 0 — wskaźnik wyrażenia tekstowego:
- 1 — prosty tekst,
- 0 — tablica tekstowa lub złożone wyrażenie tekstowe.

Bit 1 — wskaźnik obecności zmiennej w obszarze VARS:
- 1 — zmienna nie występowała,
- 0 — zmienna znajduje się w obszarze VARS.

Bit 5 — wskaźnik trybu pracy:
- 1 — tryb INPUT,
- 0 — tryb EDIT.

Bit 6 — wskaźnik oczekiwania w trybie INPUT na:
- 1 — wyrażenie arytmetyczne,
- 0 — wyrażenie tekstowe.

Bit 7 — wskaźnik rodzaju trybu INPUT:
- 1 — INPUT LINE,
- 0 — INPUT.

STRLEN (2, 5C72h, 23666). Tak jak dla ZX81.

T__ADDR (2, 5C74h, 23668). Tak jak dla ZX81.

SEED (2, 5C76h, 23670). Podstawa generatora liczb pseudolosowych. Wartość argumentu instrukcji RANDOMIZE (dla 0 jest to zawartość dwóch mniej znaczących bajtów zmiennej FRAMES) lub wartość ostatnio użytej funkcji RND pomnożona przez 65536.

FRAMES (3, 5C78h, 23672). Licznik obrazów TV. Wartość

time = PEEK 23672 + 256*
*PEEK 23673 + 65536*PEEK 23674

jest zwiększana o jeden co 0,02 s.

UDG (2, 5C7Bh, 23675). Adres początku obszaru, gdzie znajduje się generator 21 zdefiniowanych znaków graficznych. Można zdefiniować kilka generatorów i zmieniając wartość zmiennej UDG stosować kilka zestawów znaków graficznych.

COORDS (2, 5C7Dh, 23677). Współrzędne ostatnio kreślonego punktu, x = = COORDS__lo, y = COORDS__hi.

P__POSN (1, 5C7Fh, 23679). Pozycja (33 – P__POSN) ostatnio wprowadzonego znaku w buforze drukarki.

PR__CC (1, 5C80h, 23680). Tak jak dla ZX81.

ECHO__E (2, 5C82h, 23682). Kolumna (33 – ECHO__E__lo) i wiersz (24 – ECHO__E__hi) poprzedniej pozycji wydruku w systemowej części ekranu TV.

DF__CC (2, 5C84h, 23684). Tak jak dla ZX81.

DFCCL (2, 5C86h, 23686). Adres w obszarze D__FILE (część systemowa), pod którym zostanie umieszczony kolejny znak.

S__POSN (2, 5C88h, 23688). Tak jak dla ZX81.

SPOSNL (2, 5C8Ah, 23690). Kolumna (33 – SPOSNL__lo) i wiersz (24 – SPOSNL__hi) pozycji wydruku w systemowej części ekranu TV, gdzie zostanie umieszczony kolejny znak.

SCR__CT (1, 5C8Ch, 23692). Licznik wykonanych przesunięć obrazu o jeden wiersz w górę (tzw. *scroll*). Po każdorazowym przesunięciu zawartość SCR__CT jest zmniejszana o 1. Jeżeli przed zmianą wiersza SCR__CT = 1, to system zatrzymuje wykonywanie programu z zapytaniem „scroll?", po odpowiedzi pozytywnej SCR__CT otrzymuje wartość nie większą niż 18h.

ATTR__P (1, 5C8Dh, 23693). Stałe (obowiązujące dla całego ekranu) informacje o kolorach.

Bit 0...bit 2 — kod INK (0...7), (111)b oznacza też INK 8 lub INK 9.
Bit 3...bit 5 — kod PAPER (0...7), (111)b oznacza też PAPER 8 lub PAPER 9.
Bit 6 — kod BRIGHT (0,1).
Bit 7 — kod FLASH (0,1).

MASK__P (1, 5C8Eh, 23694). Stałe (obowiązujące dla całego ekranu) informacje o efektach specjalnych dotyczących kolorów.

Bit 0...bit 2 — (111)b oznacza INK 8 lub INK 9, (000)b oznacza INK (0...7).
Bit 3...bit 5 — (111)b oznacza PAPER 8 lub PAPER 9, (000)b oznacza PAPER (0...7).
Bit 6 — 1 oznacza BRIGHT 8,
0 oznacza nie BRIGHT 8.
Bit 7 — 1 oznacza FLASH 8,
0 oznacza nie FLASH 8.

ATTR__T (1, 5C8Fh, 23695). Chwilowe (obowiązujące dla danego wydruku) informacje o kolorach. Dekodowanie takie, jak w ATTR__P.

MASK__T (1, 5C90h, 23696). Chwilowe (obowiązujące dla danego wydruku) informacje o efektach specjalnych dotyczących kolorów. Dekodowanie takie, jak w MASK__P.

P__FLAG (1, 5C91h, 23697). Wskaźniki kontrolne systemu.

Bit 0 i bit 1 — kod OVER.
Bit 2 i bit 3 — kod INVERSE.
Bit 4 i bit 5 — 1 oznacza INK 9,
0 oznacza nie INK 9.
Bit 6 i bit 7 — 1 oznacza PAPER 9,
0 oznacza nie PAPER 9,

przy czym bity parzyste są wartościami chwilowymi, a nieparzyste — stałymi.

MEMBOT (30, 5C92h, 23698). Tak jak dla ZX81.

RAMTOP (2, 5CB2h, 23730). Tak jak dla ZX81.

P__RAMT (2, 5CB4h, 23732). Najwyższy (ostatni) adres dostępnej komórki w pamięci RAM (7FFFh dla ZX Spectrum 16K lub FFFFh dla ZX Spectrum 48K i ZX Spectrum +).

Dla tych, którzy chcieliby poszerzyć swoje wiadomości o systemach ZX przed następnymi odcinkami, podajemy spis wybranej literatury dotyczącej tematu.

- Logan I., O'Hara F. **„The Complete Spectrum ROM Disassembly"**, Melbourne House (Publishers) Ltd., Leighton Buzzard 1983.
- Logan I., O'Hara F. **„The Complete Timex TS1000 & Sinclair ZX81 ROM Disassembly"**, Melbourne House (Publishers) Ltd., Leighton Buzzard 1982.
- Vickers S. **„Sinclair ZX Spectrum BASIC programming"**, Sinclair Research Ltd., Cambridge 1983.
- Vickers S. **„ZX81 BASIC Programming"**, Sinclair Research Ltd., Cambridge 1981.
- Kadlof A. Broszura z serii **ABC „Komputera"** dotycząca ZX Spectrum.
- Kuryłowicz K., Madej D., Marasek K. **„Przewodnik po ZX SPECTRUM"**, WKiŁ, Warszawa 1986.

Zachęcamy do zapoznania się z wybranymi pozycjami!

M D 484

W tym artykule będzie mowa o przyszłości. Jest nią technologia arsenku galu, umożliwiająca stosowanie niskich wartości prądów progowych na złączach i uzyskanie dużej szybkości pracy układów. W 1983 roku wykonano w tej technologii eksperymentalną pamięć statyczną o pojemności 1 KB. Dwa lata później budowano już pamięci statyczne o pojemności 16 KB i układy zawierające do dwóch tysięcy bramek logicznych.

Rok 1986 zaowocował 4-bitowym mikroprocesorem wytworzonym przez firmę Mc Donnell Douglas. MD2901 wykonano w technologii JFET (ang. *Junction Field Effect Transistors*), w której bramki logiczne są wytwarzane przez bombardowanie płytki z GaAs strumieniem jonów. Szerokość bramek w mikroprocesorze wynosi 1 mikrometr, moc wydzielana w pojedynczej bramce nie przekracza 200 mikrowatów. Układ mieści się na polu o wymiarach 3,3 × 3,3 milimetra, na którym upakowano 1860 tranzystorów. Pobór mocy wynosi 135 miliwatów.

MD2901 jest funkcjonalnym odpowiednikiem stosowanego w wielu sterownikach Am2901 — ale znacznie szybszym. W ciągu sekundy wykonuje 25 milionów instrukcji maszynowych, co oznacza moc 25 MIPS. W czasie prób gotowego urządzenia stwierdzono, iż jego wydajność jest nieco niższa, niż zakładano. Winne były tworzące się na połączeniach pojemności pasożytnicze, które zwiększały czas przebiegu sygnału. Ich powstawanie zostało uwzględnione przy projektowaniu następnego mikroprocesora.

Jest nim — na jesieni '87 jeszcze nie wykonany w całości — MD 484. Zaprojektowano go jako 32-bitowy mikroprocesor RISC o wydajności około 100 MIPS. Architekturę MD 484 wzorowano na istniejącym już mikroprocesorze MIPS (nie mylić ze skrótem jednostki!) wykonanym w Stanford. Umożliwiono pobranie danej lub instrukcji w ciągu 5 nanosekund i współpracę z maksymalnie ośmioma koprocesorami arytmetycznymi.

Mikroprocesor ma posiadać trzy magistrale: instrukcji, danych i adresową. Stosowanie tej samej magistrali do generowania adresów danych i rozkazów nie stworzy wąskiego gardła dla obliczeń. Będzie ona wykorzystywana tylko przy pobieraniu danych, wykonywaniu instrukcji skoku i wydawaniu komend koprocesorom. W komputerach wykorzystujących MD 484 pamięć instrukcji zostanie wyposażona we własny licznik rozkazów i układy zajmujące się automatycznym dostarczaniem rozkazów jednostce centralnej. Wykonanie skoku będzie wymagało jedynie podania przez mikroprocesor adresu, który zostanie wpisany do zewnętrznego licznika rozkazów.

Mikroprocesor ma zawierać 17 rejestrów 32-bitowych ogólnego przeznaczenia. Podczas jego projektowania przewidziano miejsce dla ośmiu dalszych rejestrów, które mogą być dodane w miarę rozwoju technologii GaAs. CPU będzie też wyposażony w 24-bitowy licznik rozkazów, rejestr statusu i kilka rejestrów ze znacznikami. Kod operacji zajmuje sześć najbardziej znaczących bitów 32-bitowego słowa, pozostałe przeznaczono na operandy. Analizy wykazały, że w programie 91% instrukcji będzie wykonywanych w pojedynczym cyklu zegarowym. Reszta zostanie automatycznie przetłumaczona na ciąg instrukcji prostych.

Mikroprocesor będzie zawierał 23 176 tranzystorów i 10 384 rezystory umieszczone na powierzchni 6,5 × 11 mm. Przewidywana moc pobierana przez układ wyniesie 4—6 watów. Podczas projektowania CPU stosowano zasadę czasowej ścieżki krytycznej. Podsystemy mogące wydłużyć czas cyklu zegarowego konstruowano tak, by zmniejszyć opóźnienia sygnału — bez troski o liczbę tranzystorów czy pobór mocy. W pozostałych ilość elementów i spodziewane straty mocy optymalizowano.

Mc Donnell Douglas buduje prototyp mikroprocesora w trzech etapach. Pierwszy stanowił układ Demo I zawierający około 5000 tranzystorów — Jednostkę Arytmetyczno-Logiczną. Następnie wytworzono i przetestowano Demo II złożony z ponad 17 tysięcy tranzystorów. Pozostał już tylko krok do otrzymania kompletnego mikroprocesora.

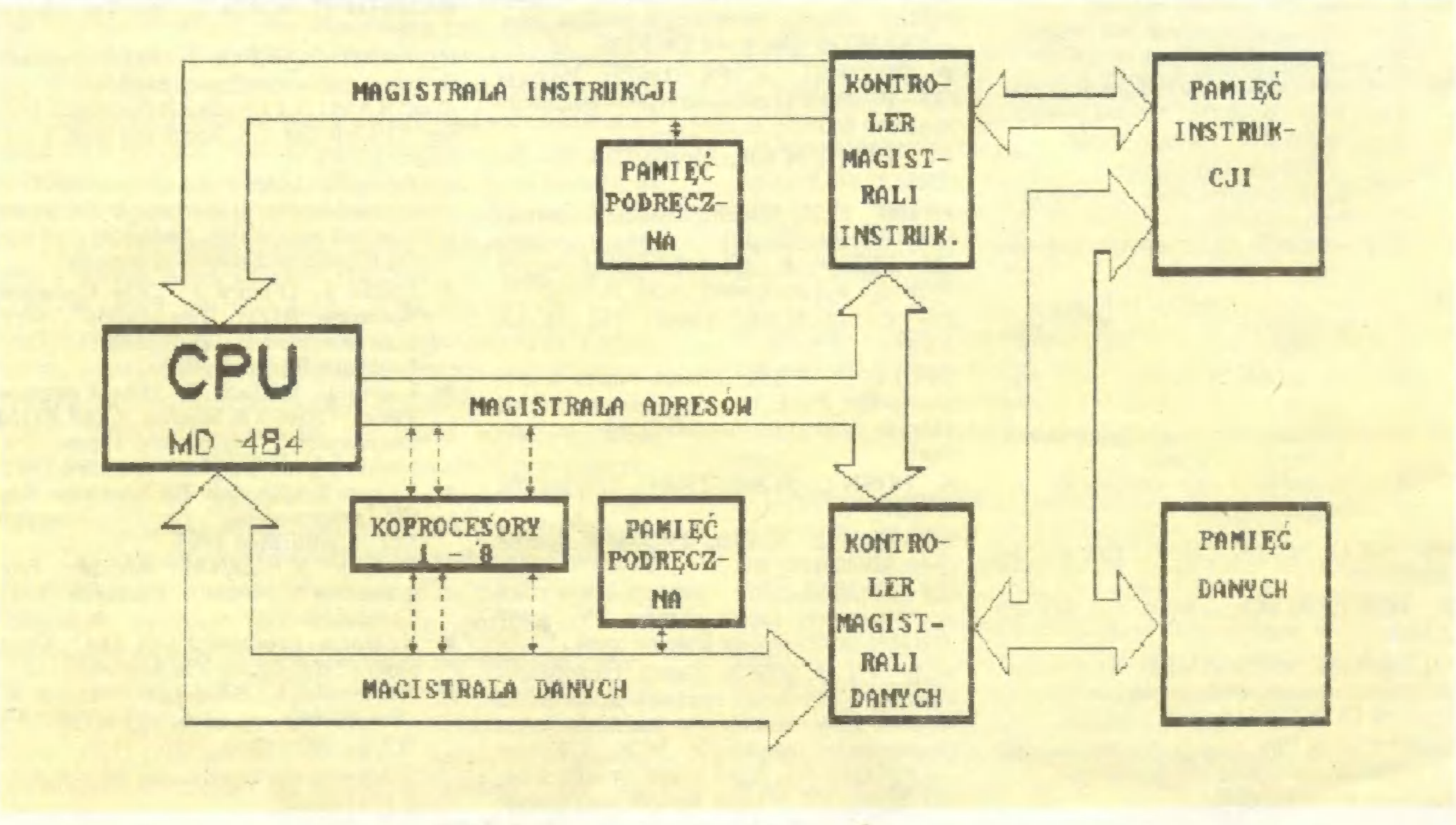
Koprocesory arytmetyczne współpracujące z MD 484 zostaną również wytworzone z arsenku galu, będą zawierały ponad 26 tysięcy tranzystorów. Będą operować na danych pojedynczej (32 bity) i podwójnej (64 bity) precyzji zapisanych w formacie IEEE. Wyniki operacji będą zapamiętywane z większą dokładnością, niż dane wejściowe — dla zmniejszenia błędów zaokrągleń. Będzie możliwe zaprogramowanie zaokrąglenia wykonywanego po całym ciągu instrukcji, w czasie egzekwowania których argumenty pozostaną w formacie rozszerzonym.

Przy planowanej szybkości wykonywania rozkazów, niewielkie różnice czasu przechodzenia sygnałów przez bramki logiczne procesora i koprocesora mogą zakłócić synchronizację obliczeń. Dlatego zarówno dla CPU, jak też dla koprocesora przewidziano identyczne ścieżki wejścia/wyjścia i bufory. Obydwie jednostki będą taktowane tym samym, zewnętrznym zegarem (w IBM PC/AT zegary procesora i koprocesora mogą mieć inną częstotliwość pracy). Dzięki temu zostanie nie tylko ulepszona synchronizacja, ale też umożliwiona praca krokowa całego systemu podczas uruchamiania programu.

Wychwytywanie rozkazów dla koprocesora przez CPU i ich przekazywanie magistralą adresową wyeliminuje konieczność dołączania do koprocesorów osobnych magistral instrukcji i uprości ich budowę.

W czasie prób z układami zbudowanymi z arsenku galu stwierdzono, że opóźnienie sygnału na pojedynczej bramce jest tak małe, iż o szybkości pracy systemu decyduje czas przekazywania sygnałów pomiędzy układami logicznymi. Jest on tym mniej znaczący, im więcej bramek mieści się w obszarze jednej struktury. Oznacza to, że będzie można częściej wykorzystywać technologię GaAs wtedy, gdy inżynierowie opracują metody budowania układów VLSI z arsenku galu.

wg M. and M. 8/87
opracował Adam Nowicki



PRZYPRAWY DO LOTUSA

Pakiet **1-2-3** firmy Lotus ustanowił wśród elektronicznych arkuszy obliczeniowych standard wciąż trudny do pokonania. Nic więc dziwnego, że wielu producentów oprogramowania zrezygnowało z prób wprowadzania na rynek konkurencyjnych produktów. Postanowili oni natomiast wykorzystać popularność 1-2-3 poprzez uzupełnianie i rozszerzanie jego funkcji. W ten sposób pojawiło się na rynku wiele programów, określanych jako ,,dodatki do Lotusa".

Typowy program-dodatek ma postać nakładki, wczytywanej do pamięci wraz z samym 1-2-3 i działającej na zasadzie rezydującego programu usługowego — takiego, jak np. SideKick. Specyficzną cechą ,,dodatków" jest bezpośredni dostęp do danych w arkuszu i zgodna z 1-2-3 forma komunikacji z użytkownikiem.

Dodatkiem ułatwiającym wprowadzanie danych do arkusza jest program **D.a.v.e** firmy Goldata Computer Services. Użytkownik może zdefiniować dowolne formularze, każdy składający się z maksymalnie 10 ekranów po 256 pól. Wpisywanie danych polega na prostym wypełnianiu wolnych pól formularza, a wpisane dane są wprowadzane bezpośrednio do arkusza 1-2-3.

D.a.v.e uwalnia użytkownika od kłopotów z błędnie wprowadzonymi danymi — dzięki bogatym możliwościom kontroli. Można zdefiniować granice, w jakich muszą się zawierać wartości, podać listę dopuszczalnych możliwości lub utworzyć referencyjny plik ASCII. Można nawet nakazać sprawdzanie, czy każda pozycja kolumny ma unikalną wartość. Kontrola poprawności danych obejmuje także istniejące już arkusze — wystarczy padać wymagane kryteria, a D.a.v.e wskaże podświetleniem pola, których zawartość budzi podejrzenia. Pola te mogą być następnie poprawiane z wykorzystaniem wygodnego edytora programu D.a.v.e Można też wydrukować raport o wykrytych usterkach.

D.a.v.e rozszerza możliwości wydruku danych z arkusza — każdy formularz zdefiniowany do wprowadzania danych może posłużyć jako wzorzec wydruku. Takie podejście pozwala jednocześnie zwiększyć pojemność arkusza — wszelkie opisy pól umieścić na szablonac[illegible] formularzy, pozostawiając komórki ark[illegible] sza na dane liczbowe.

Program D.a.v.e współpracuje z [illegible] sją 1-2-3 Release 2 (lub późniejs[illegible] zajmuje 64 KB pamięci komputera [illegible] sztuje 149,95 dolarów.

Kolejnym producentem, który po[illegible] wił uzupełnić swoimi programami [illegible] Lotusa, tym razem w dziedzinie p[illegible] twarzania tekstu, była firma Funk [illegible] ware. Oferowany przez nią pro[illegible] **InWord** jest profesjonalnym edyt[illegible] tekstu, pracującym w środowisku 1-[illegible]

InWord zawiera wszystkie typowe [illegible] kcje edytorskie dostępne w samodz[illegible] nych procesorach tekstowych. Moż[illegible] wprowadzać fragmenty arkusza d[illegible] tekstu, a wybrane partie tekstu umieszczać w polach arkusza. Szczególnie przydatną cechą programu InWord są tzw. żywe połączenia — w opracowywanym dokumencie można umieścić odnośniki do wybranych pól arkusza, a każda zmiana zawartości tych pól będzie natychmiast odwzorowana w tekście. Można także korzystać z bazy danych zawartej w arkuszu przy seryjnych wydrukach podobnej treści (funkcja *mailmerge*).

InWord wykorzystuje pełne możliwości drukarek — od typowych 9-igłowych po laserowe — do tworzenia profesjonalnie wyglądających wydruków. Można używać kursywy, druku wytłuszczonego i podkreślonego, mieszając dowolnie kroje i wielkości czcionek.

Z myślą o użytkownikach skomplikowanych arkuszy firma Funk opracowała program **Noteworthy**, przeznaczony do opisywania i dokumentowania zawartości arkusza. Dzięki niemu nie trzeba się zastanawiać, co kryje się za treścią danej komórki — wystarczy przycisnąć ,,gorący klawisz" i wywołać na ekran notatkę opisującą wskazywane pole. Notatki — liczące do 8000 znaków każda — mogą być tworzone za pomocą wbudowanego edytora programu, w dowolnie zdefiniowanym (co do rozmiarów i położenia) okienku na ekranie. Można przenosić zapiski z arkusza do notatki i odwrotnie, a także pomiędzy notatkami. W czasie pracy Noteworthy podpowiada, które komórki arkusza są opisane — po wskazaniu takiej komórki na ekranie zapala się wskaźnik notatki.

Ofertę firmy Funk Software uzupełnia program **Sideways** — generator wielkoformatowych wydruków. Dzięki niemu użytkownik arkusza liczącego wiele kolumn nie musi się liczyć z szerokością papieru w drukarce. Sideways pozwala drukować wzdłuż wstęgi papieru, wykorzystując dziewięć różnych wielkości czcionek i stosując druk wytłuszczony, podkreślony i poszerzony. Program pozwala definiować marginesy i format stron, dodawać opis[illegible] jać perf[illegible]

Programy firmy Funk współpracują z pakietami 1-2-3 Release 2 (Sideways również z wcześniejszymi wersjami) oraz Symphony. Ich ceny mieszczą się w zakresie poniżej 100 dolarów za program.

Kompleksową oprawę Lotusa 1-2-3 proponuje też Turner Hall Publishing. W ofercie znajdują się programy podobne do opisanych powyżej — integralny edytor tekstowy **4WORD** i program do opisywania pól arkusza **Note-It Plus**. Popularną już od pewnego czasu funkcją jest kompresja danych z arkusza. Program **SQZ!** pozwala zredukować objętość pliku zajętego przez arkusz nawet o 95% (!), uwalniając znaczną przestrzeń dysku i nieporównywalnie przyspieszając wczytywanie arkusza do pamięci.

Kontrolę poprawności danych w arkuszu zapewnia **The Cambridge Spreadsheet Analyst** — program obecny już w większości liczących się biur Ameryki. The Analyst potrafi wyszukać 25 rodzajów błędów w arkuszu, badając nie tylko wartości pól, ale i powiązania między nimi. Wbudowany The Spreadsheet Comparator pozwala kontrolować różne wersje tego samego arkusza i nadzorować poprawność arkuszy rozproszonych. The Analyst zawiera też narzędzia do analizy i korekty makrodefinicji. Można też zadbać o poprawność językową arkusza. Zapewnia ją program **Spellin!**, sprawdzający pisownię wyrazów wpisanych w pola arkusza.

Najdroższe z programów Turner Hall — 4WORD i The Cambridge Spreadsheet Analyst — kosztują po 99,95 dolarów. Programy te są już dostępne w wersji dla komputerów PS/2 — na dyskietkach 3,5--calowych.

Obserwatorzy rynku uznają dodatki do popularnych pakietów za szczególnie obiecującą gałąź oprogramowania do zastosowań profesjonalnych. Nie chodzi jednak tylko o komfort pracy, lecz również o bezpieczeństwo prowadzonych interesów — nietrudno wyobrazić sobie straty, jakie może spowodować błąd w arkuszu opisującym wielomilionowe transakcje lub błędna interpretacja za[illegible] [illegible]órki takiego arkusza.

[illegible]



Cena zł 400,—